Multicriteria Mission Route Planning Using a Parallel A* Search

THESIS

Michael S. Gudaitis
Captain, United States Air Force

AFIT/GCS/ENG/94D-05

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/94D-05

Multicriteria Mission Route Planning Using a Parallel A* Search

THESIS

Michael S. Gudaitis
Captain, United States Air Force

AFIT/GCS/ENG/94D-05

AFIT/GCS/ENG/94D-05

Multicriteria Mission Route Planning Using a Parallel A* Search

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Michael S. Gudaitis, B.S., M.S.E.E.

Captain, United States Air Force

13 December 1994

Acknowledgements

I owe a great deal of thanks to all those people who provided guidance, assistance, and support to me in this research. My advisor, Gary Lamont, constantly pushed me towards greater achievement, and it was through his encouragement that I made the breakthroughs in my research. My committee members, Yupo Chan, Andrew Terzuoli, Eugene (Doc) Santos, and Lt Col Bill Hobart provided valuable input in analyzing my results and refining this document. My coworkers in the parallel lab, Capt's Jim Hiller, Jay Graham, and George Gates, provided a sounding board for my ideas, and helped relieve the monotony and drudgery of many late-night hours in the parallel lab working on my research.

I especially thank my family for providing the hugs and reassurance when I needed them most, and for sharing my disappointments and successes. I am indebted to my wife Jean who is my greatest fan, and has provided me with love and understanding. I could not have done this work without her. My children, Edward and Catherine, have been a great joy to me, and have made this all seem worthwhile.

My greatest thanks go to God for blessing me abundantly, and allowing me to succeed. Through Him, all things are possible, and my success here at AFIT attests to that. I place all my trust in Him.

Michael S. Gudaitis

Table of Contents

Page

List of Figures

List of Tables

AFIT/GCS/ENG/94D-05

Abstract

Mission Route Planning is a critical element of pre-flight planning for military pilots and navigators. The planners must assemble information on the target, terrain, threats, weather, weapons, and more, to select the "best" flight path for an aircraft to complete its mission and return safely. In wartime situations, this planning must be completed in the time between missions, typically three hours or less. An automated route planner capable of selecting "optimal flight paths" in less than three hours would free pilots and navigators to concentrate on other apects of the mission. This problem is known as the multicriteria Mission Route Planning (MRP) problem.

The Mission Route Planning (MRP) problem falls into the general class of multicriteria path search problems. Multiple criteria are evaluated to select an optimal aircraft mission route through a hostile environment. This problem is shown to be NP-Complete. However, the problem complexity is reduced to $O(n^2)$ by combining the multiple criteria into a single cost function. For the bicriteria MRP problem, the criteria for distance travelled and radar exposure are combined into a single cost function for route evaluation. Radar calculations are performed dynamically using a simplified monostatic radar model. The terrain is implemented as a three-dimensional array of XYZ values representing latitude, longitude, and elevation. The A* search algorithm is selected for its efficiency over other methods, and a parallel implementation is developed and tested. The heuristic function used in A* reduces the size of n in $O(n^2)$ to give faster execution times than other methods. The A* algorithm incorporated a unique combination of distributed OPEN lists with a global CLOSED list strategy which produces fast execution times on the Intel Paragon. Test cases for scenarios with 15 radars took less than 5 minutes with 16 processors. Analysis shows performance to be scalable. When compared to a fast sequential A* implementation, the parallel A* version showed linear speedup. Tests were performed on the iPSC/2, iPSC/860, and Paragon. Measured performance is far superior to previous efforts.

Multicriteria Mission Route Planning

Using a Parallel A* Search

I. Introduction

1.1 Problem Statement

Mission route planning (MRP) is the process of choosing the "best" route for one or more vehicles to travel to and from one or more targets. Although it has applications to both air and ground vehicles, this research is concerned with air vehicles. Route selection for air vehicles is affected by many criteria, such as distance travelled, radar threat, weather, and more. Few automated tools exist that plan the entire route. There are tools that aid in the collection and formatting of the data, but, for the most part, route selection is still a manual process. This research is directed at finding an automated system for MRP which is effective (finds the "best" route) and efficient (finds the route in real time). The system developed should be extensible to handle multiple targets. The ultimate goal is a real-time system installed on an aircraft and capable of modifying the pre-planned route in flight to adapt to a change in criteria. Use of a parallel computer is not a goal, but a means to obtain greater time efficiency.

1.2 Background

Mission Route Planning (MRP) for aircraft is defined as a search to find the "best" flight path from a starting point through defended terrain to a target or set of targets. How is the "best" route determined when there are many conflicting criteria which must be considered in the route selection? For example, the shortest path to a defended target is likely to be one with high exposure to radar. Likewise, minimizing radar exposure would not, in general, yield the shortest route. In the past, the choice of a "best" route relied on the experience and judgment of the pilot or navigator planning the route. Factors in choosing the route included (from (6)):

Distance travelled.
Way points.
Type armament delivered to target.
Number of planes in the sortie.
Refueling points.
Threat of enemy aircraft.
Ground to air missile batteries.
Detection avoidance (radar, communication, visual, etc.)
Presence of friendly and hostile jammers.
Terrain.
Aircraft radar cross-section.
Minimum and maximum altitude.
Day or night mission.
How many targets, and in what order.
Time constraints.
Weather.

The person planning the route determines the importance of the criteria in evaluating the route selection and consciously or unconsciously assigns a weight to the criteria. The criteria weights are combined to form an additive cost function to evaluate a candidate route. The route chosen would then be the one with the highest (or lowest, depending on viewpoint) total weight. This would determine the "best" solution[1]. An additional constraint on planners is that typically they have only a few hours to plan the route before the mission is scheduled to begin.

It is apparent that this process is highly subjective, and therefore does not usually yield consistent results. The objective of this investigation is to develop a step by step procedure (algorithm) for selecting the route and implementing this on a parallel computer. One of the major hurdles to performing this on a computer is the computational complexity of determining a "best" path with many conflicting criteria. This research investigation shows that MRP is a three-dimensional, multi-criteria path search, which is an NP-Complete problem. This means that the run time for

[1]Some people refer to this "best" solution as the "optimal" solution. However, *optimal* refers to minimizing (or maximizing) an evaluation function that is a linear or non-linear combination of all the criteria. Several "optimal" solutions may be obtained by varying the route criteria of the evaluation function and selecting the "best" from the set of "optimal" solutions obtained from each combination of the criteria. The importance (or weight) assigned to each criteria is often arbitrary, subjective, or the result of trial and error. Therefore, the term *best* is used here instead of *optimal.*

a general MRP algorithm to handle all inputs would grow exponentially (or faster) with a linear increase in the input. However, for the case of two criteria, it can be shown that polynomial time algorithms exist. This research focuses on finding solutions for the bicriteria Mission Route Planning (MRP) problem in polynomial time.

To accomplish this goal, a weighting function is developed to determine the weight of each chosen path in the route and then develop an algorithm which selects the path of minimum weight. The weighting function must take into account all the conflicting criteria and assign a number (or weight) to each path. The assignment of different weights (signifying level of importance) to the route evaluation criteria is investigated to determine how the weighting affects the selection of the route. The goal is to develop an algorithm which provides an optimal solution to the specific evaluation function in one hour or less for realistic input data. Realistic input data is defined as the actual data (terrain maps, target information, etc.) used to select the route.

1.2.1 Route Selection Criteria.

As discussed previously, many criteria are evaluated to select the best route. Two criteria considered in this research are exposure to radar and distance travelled. These two criteria were selected because they represent the two categories, distance and threat, that include most of the other criteria. For example, time on target, speed, and fuel consumption can be derived from distance travelled. The other category, mission threats, include radar exposure, adverse weather conditions, radio and visual detection, and enemy aircraft. The two chosen criteria are covered in more detail in the following sections:

1.2.1.1 Radar Threat.

One of the criteria for route selection is probability of detection. Two basic methods are used—static and dynamic. In (50), the radar threat was pre-calculated and overlaid on each region of the search space. Orientation of the aircraft was not considered. The search tried to minimize flying through regions of radar coverage. This is the static method. In the dynamic method, computing probability of detection at each point in the route is very complicated. It depends on such factors as the radar cross-section of the aircraft, the aircraft's orientation (i.e.

azimuth, elevation, and degree of bank (or roll)), and the distance of the aircraft to the radar transmitter and receiver. The more aircraft orientations considered, the more complex the algorithm becomes, and the longer it takes to find a solution. There are various simplifications which can be made here. The most common is to treat the aircraft as a sphere so that the radar cross section appears the same from any angle. While this is OK for a general approach, it obviously does not take into account the advantages of stealth aircraft (i.e. aircraft with low-observable radar cross-sections). The approach taken in (29) used the sphere model for the aircraft radar cross-section. This essentially treats the aircraft as a moving point. Improvements to this model are investigated and its implications to complexity are discussed.

In addition to probability of detection, it is important not only to know the total exposure to radar, but also the longest uninterruptible exposure to radar. For example, one 60-second exposure is much worse than six 10-second exposures with no detection between the 10-second segments. This is especially critical with anti-aircraft missile sites, where it can take up to 60 seconds to lock onto a target, but only a few seconds to initially detect it. Therefore, this research investigates both of these aspects of radar exposure, with the goal of minimizing either, or both, these radar criteria.

1.2.1.2 Distance Travelled. The selected route must be within the range of the aircraft. The minimim route distance is the straight line distance "as the crow flies" from the starting point to the target. The maximum distance is the maximum range of the aircraft configured for its mission. With a computer, the simplest representation for the terrain is a three-dimensional grid map of equal grid sizes defined in terms of latitude, longitude, and elevation. The route distance is computed by adding up the distance between all the grid cells that the selected route passes through. The computer may check all grid cells to find a best route. A disadvantage of this approach is the number of grid cells required to provide a given resolution of the search space. For example, a 1000^3 meter search area with required resolution of 10 meters in latitude, longitude, and elevation would generate 1,000,000 grid cells. A resolution of 100 meters would need only 1000 cells. This can have a significant effect on the run-time of any algorithm we choose. However, the discrete points provide the simplest method for dynamically calculating radar exposure of the aircraft as it flies its irregular route to the target.

1.3 Scope and Objectives

This research investigation builds upon previous research at AFIT, with Droddy's work (29) being the most recent and most directly applicable. Droddy implemented dynamic radar calculations, improved the route evaluation function, and modularized the software implementation. These enhancements provided significant performance gains over Grimm's work (50). However, inefficiencies existed in Droddy's software which caused it to fail to meet its real-time objectives for some scenarios. Therefore, a more thorough mathematical analysis of the problem was performed in this research which led to the design of a more efficient algorithm. The new design corrected the inefficiencies of Droddy's work, and experiments validated that the real-time objective was met for all scenarios tested. Specifically, this research consisted of the following tasks:

- Perform a thorough mathematical analysis of the problem complexity, and determine if MRP can be mapped to other general problems for which solution algorithms already exist.
- Determine which algorithms can be used to solve MRP, and select one for implementation.
- Improve the stopping criteria for Droddy's parallel implementation of the A* (pronounced A-star) algorithm.
- Improve the representation of the radar cross-section model of the general aircraft and include this in the algorithm. Determine the added complexity for calculating probability of radar detection based on this new model and aircraft orientation.
- Investigate new data structures such as a quadtree representation of the 3-D search space. This may give the desired grid resolution with fewer total nodes to search than a standard square grid representation.
- Derive a complexity equation that determines the time complexity based on the number of route criteria used. As a minimum, two criteria are considered – distance travelled, and probability of detection. Other criteria that may be added are variable speeds, weather, and time constraints.
- Investigate the complexity of generalizing MRP to multiple aircraft with multiple targets.
- Investigate performance issues (speed, scalability, efficiency) on massively parallel processors.

1.4 Approach

First, the problem is analyzed to determine which general problem models can be mapped to MRP (and vice-versa). The problem is formally specified using 0/1 integer notation, and the time and space complexity determined.

Second, algorithms to solve the problem are investigated, and an approach chosen. A literature search yielded papers which describe research applicable to MRP (18, 48, 60, 76, 84, 108, 117). Most are sequential approaches using various graph search methods such as A* (18, 48, 60, 76), dynamic programming (60, 84), and potential fields (117). Some use quadtrees (18, 108) to represent the search space. The research behind these papers are investigated to determine their applicability to the general MRP problem.

Third, an improved radar model is developed and added to the algorithm. The improved model includes a minimum detectable angle, an increased weighting for consecutive radar exposure.

Fourth, a parallel program is encoded to implement the algorithm on Intel parallel computers: iPSC/2, iPSC/860, and Paragon. Existing code is used as much as possible. Various input files are developed to test the performance of the programs. Debugging and fine-tuning of the programs is performed as necessary. Tests included measurements of parallel performance issues (speedup, efficiency, scalability).

Lastly, the results of the experiments are analyzed to determine how well the objectives were met. The results are summarized and the recommendations are given concerning areas for continued research.

1.5 Thesis Overview

This chapter provides an introduction to the mission routing problem and discusses the scope of this research. The remainder of the thesis is composed of six chapters. Chapter II presents an overview of the general MRP problem, the computational models used, and the complexity of the problem. Chapter III discusses the algorithms used to solve routing problems. Chapters IV and V give the software design of the algorithm developed in this research, from high-level to low-level design and implementation. Chapter VI reports the results of experiments conducted on the previous design compared to the results from the new design developed in this research. Conclusions and recommendations for future research are given in the final chapter, Chapter VII.

II. Overview of General Mission Route Planning (MRP) Problem

2.1 Introduction

This chapter presents an overview of the MRP problem. It begins with a description of the problem, presents the computer models used in solution techniques, and concludes with a discussion of the complexity of the problem.

2.2 Description of General MRP

The general Mission Route Planning (MRP) Problem is the process of selecting a flight path for aircraft to fly from a starting point through defended terrain to target(s), and return to a safe destination[1] (6, 52). The aircraft moves to and from the target(s) through three-dimensional space avoiding solid obstacles (terrain), and avoiding threat areas or moving through them at increased risk to the mission. Path criteria are evaluated in order of importance to the mission, and a route is selected which satisfies the criteria. Many criteria are used in the evaluation of selected paths. Some criteria must be strictly adhered to. For example, maximum range of the aircraft constrains the maximum path distance; maximum speed limits the earliest time on target. Other criteria are more subjective. For example, what probability of detection is considered acceptable? Satisfactory solutions form the set of all solutions that meet mission route criteria. An optimal solution is one that optimizes the route criteria. Optimality is determined by a weighted cost function of multiple criteria (objectives) which describes the route. The higher the cost, the greater the risk. The objective of a mission route planner is to select the route that minimizes the risk to the mission. For the general MRP problem, a route planner typically must plan for multiple aircraft against multiple targets. The number of criteria that may be used to evaluate flight routes were described previously in Section 1.2.

[1]For unpiloted aircraft such as a cruise missile, the final destination is the target, so there is no return portion.

In this research effort, the general problem is constrained to one aircraft against one target. In terms of aircraft survivability, only two criteria are considered for route selection: minimum total distance travelled, and greatest avoidance of detection. One objective of this research is to show that the MRP problem is a three-dimensional, multi-criteria path search, which is an NP-Complete problem. This implies that current algorithms which solve the general problem take exponentially longer times with linear increases in the input size (40). Thus, an exponential time algorithm could take days to complete for a given input, compared to minutes or seconds to complete with an algorithm of polynomial time complexity. A goal of "reasonable" time is 45 minutes[2] at worst for very large input. To accomplish this goal with a computer, the real-world environment of the problem must be mapped into discrete mathematical models and data structures that a computer can manipulate to find a solution. An algorithm must be selected and implemented to provide an effective and efficient method for evaluating the various models. Finally, the implementation is tested and the results analyzed to assess the effectiveness and efficiency of the solution technique. The first step towards finding a solution technique for MRP is developing computer models to represent the real-world.

2.3 Discrete Models of the Real World

Models must be developed in order for a computer to meet the goals of an optimal mission route planner, which are to determine the shortest flight route that minimizes detection (radar), and do it in one hour or less for real mission scenarios. To accomplish this, the discrete models that must be developed to represent the real world are as follows:

- Digital representation of the terrain (maps, grids, etc.).
- Radar detection and Radar Cross Section (RCS).
- Model of aircraft movement.
- Cost function to evaluate the criteria for selecting the "optimal" path.

[2]It takes approximately 45 minutes to prepare a combat aircraft to return to battle, but it currently takes hours to re-plan a mission (52, 53).

2.3.1 Model of Terrain/Search space.

If we could develop a general equation of a line that includes the least-cost path as a line segment, then we could perform a direct evaluation to compute the least-cost path between any two given points. However, no such general equation exists. Therefore, the search space must be discretized such that the least-cost path can be computed by interconnecting subpaths from a starting point to a goal, and evaluating the "optimality" of the path when the goal is reached. For MRP, a three-dimensional model is required to represent the terrain and flight space for an aircraft. One model for the terrain is a standard three-dimensional grid of x, y, and z coordinates that represent latitude, longitude, and elevation, respectively. Another model that could be used is a polygon model. The advantages and disadvantages of these models are discussed in the following sections.

2.3.1.1 **Polygon Model:**

In some applications of robot path planning, the obstacles (*i.e.* terrain) are represented by polygons (87) and the vertices of the polygons are stored in a visibility graph. The resolution depends on the number of segments used to represent the obstacles, as shown in Figure 2.1. To compute the route around obstacles, only the locations of the vertices are needed. Ray tracing is used to avoid intersection of the path with any obstacle vertex before the goal is reached. An optimal route is one that traces a path from start to goal without intersecting any vertices. The complexity of this approach depends on the number of obstacles, and the number of segments used to represent the obstacles. Representing this approach in three dimensions greatly increases the complexity of the polygonal representation and number of possible intersection points (vertices)[3]. This, in turn, increases the complexity of computing routes that do not intersect with an obstacle vertice. Another disadvantage to this approach is that it cannot adequately model radar detection. The ability of radar to detect an aircraft varies greatly with the position (altitude, orientation) of the aircraft with respect to the radar. A polygon model of the terrain does not capture this information, and therefore is unsuitable for accurate radar calculations.

[3]Intersection calculations dominate the run time of ray tracers (47).

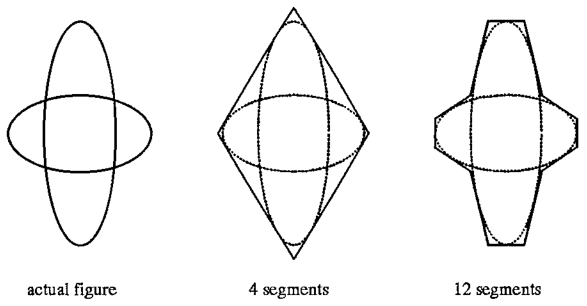

Figure 2.1 Polygon Model for Terrain Obstacles in 2 Dimensions.

2.3.1.2 **Standard Grid Model:** The simplest model that discretizes the search area is a standard 3-D grid where each grid cell represents a cube volume of terrain or sky. An advantage of this representation is its simplicity — a 3-D array can be used as the data structure. Each dimension of the array represents a physical dimension of space, therefore the entire search area is explicitly represented by discrete points. The discrete points are required to calculate radar exposure along the flight path. The amount of radar exposure varies as an aircraft moves towards or away from radar. This variation can be characterized by calculating exposure at the discrete points. A polygonal terrain model cannot do this. A disadvantage of a standard grid is that the number of grid cells needed for a given resolution is $O(d^3)$, where d is one dimension (*e.g.* latitude) of the terrain representation. Therefore, doubling the resolution of the grid would multiply the number of grid cells by a factor of $2^3 = 8$. However, if the dimension of Z-axis (elevation) is defined separately from the X-axis (latitude) and Y-axis (longitude), the resolution in height (Z-axis) can be changed in $O(z)$, and the resolution in the plane of terrain can be changed with only $O(xy) = O(d^2)$ increase in grid cells. This is the format used by the Defense Mapping Agency (DMA) for its Digital Terrain & Elevation Data (DTED) which provides digital terrain information for any area of the globe (86:182-183). Also, the operating environment of aircraft is typically measured in feet or meters for elevation (maximum operating ceiling) and miles or kilometers for distance (XY plane of latitude and longitude). The advantages of this model outweigh the disadvantages. Therefore, the grid representation is the model selected for this research because of its simplicity, and because of its advantage over other terrain models in dynamic calculations of radar detection with multiple radar cross-sections (RCSs)[4]. The next section describes a method used by some path planners to reduce the number of explicit gridpoints that must be searched.

2.3.1.3 **Quadtree Grid Model:** Some route planning applications that use a grid model have incorporated a quadtree data structure to reduce the number of grid cells needed to represent the search area. In (18), quadtrees were used to store terrain information for autonomous underwater vehicles, and in (108, 109) quadtrees held terrain information for a helicopter route planner. These applications assume that vehicle threats can be assigned to grid cells independent of the vehicle position. Areas of similar threat are then grouped together in a quadtree structure, as in Figure 2.2. The advantage to this approach is that the overall number of grid cells needed to represent a given area of terrain may be reduced. This has the effect of reducing the size of "n" for an $O(n^c)$ or $O(c^n)$ complexity (c is a constant). This in turn may reduce the overall execution time. For example, in the Figure 2.2, the quadtree model represents in 102 cells what would take 384 cells using a standard grid. The size of the grid cell varies depending on the amount of "interesting" terrain in the cell and the resolution needed to capture the important features. Mountainous areas would be represented by smaller grid cells for high resolution, while plains and open sky could be represented by much larger grid cells. A disadvantage of this approach is that it does not allow dynamic calculation of radar detection. The radar detection calculations would be based on the position of the aircraft in the center of the larger square, so there would be error in both the range and aspect angle of the aircraft with respect to the radar. In addition, quadtrees do not provide much reduction in the number of grid cells for scenarios with a wide variation in terrain and number of threats. Another disadvantage is that quadtrees require a more complicated algorithm to determine the nearest neighbor to find the next point in the path. Therefore, quadtree

[4]RCS to be discussed in Section 2.3.2.1.

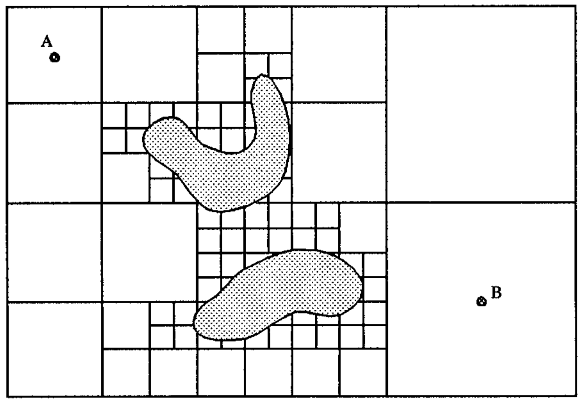

Figure 2.2 Quadtree Representation of a Planar Grid. The shaded areas represent obstacles. The goal is to get from A to B. Quadtrees allow you to concentrate the finest resolution around the areas of interest, while combining the general areas into larger grids.

structures, in general, are not useful for the fixed-wing aircraft MRP that employs dynamic radar calculations.

2.3.1.4 **Digitization Bias:** When using a grid structure, the least-cost path from starting point to goal may not be unique. This is called digitization bias (86:176,183). The shortest distance between two points in Euclidean space is a straight line. However, the shortest path in a finite grid representation is the sum of interconnecting line segments between nearest neighbor points as shown in Figure 2.3. With a grid structure, several paths with the same "cost" (*e.g.* distance) may exist. This bias is unavoidable in a grid model of the search space. Only paths starting and ending on the same axis or diagonal are unique. Because of this digitization bias, an MRP algorithm must have a method to distinguish between those "optimal" paths which are actually the same path but with digitized bias.

2.3.1.5 **Representation of the Earth:** The earth is spherical in shape, and therefore a realistic approach to route planning requires the use of curvilinear coordinates (latitude and longitude), and trigonometric formulas to calculate arc lengths between points. However, for short distances, the earth can be approximated by a flat plane. This is the approach used in this

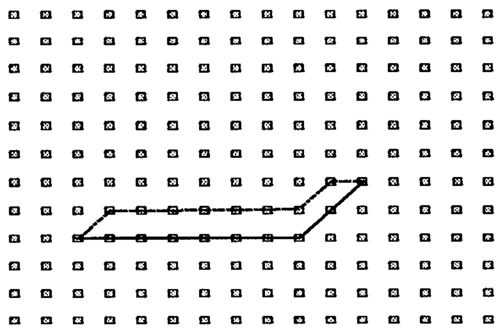

Figure 2.3 Digitization Bias: Multiple paths of equal distance resulting from grid representation of search space.

research. This approach simplifies the straight-line calculations between points. The method using curvilinear coordinates (latitude and longitude) is left as a future implementation.

2.3.2 Radar Model. The study of radar covers topics in electromagnetics, radio communication, statistics and probability, information theory, and other specialized areas. To address this area in sufficient detail would take several textbooks. It is not the intent of this research to cover all the details of radar principles. The reader is referred to any number of good texts on radar. A general introduction to radar is provided in (112). A more detailed discussion is given in the **Radar Handbook** (106). Only the details needed to develop a radar computer model are summarized in the following paragraphs:

2.3.2.1 Radar Equation. Consider an "isotropic" radar source transmitting a radar pulse of peak power P_t. An isotropic transmitter radiates the radar signal outward in waves equally in all directions. An analogy in two dimensions is the ripples formed by dropping a pebble into a pond. The radar wavefront forms a surface of a sphere of radius R with the radar energy equally distributed over the surface. The radar pulse can be concentrated and focused in a certain direction by an antenna. The ratio of the focused energy over the isotropic energy is called the gain G_t of the transmitting antenna. The energy wave intercepts a target (aircraft) at range R from the radar. The amount of energy reflected back to the radar receiver depends upon the radar cross-section σ of the target (aircraft). Reception of the reflected signal is dependent upon the strength of the signal S, the characteristics of the radar receiver, and its location with respect to the transmitter and target. The signal-to-noise ratio S/N at the receiver is used to determine detection of an aircraft. The S/N is defined by the following general equation (106:25.6):

$$\frac{S}{N} = \frac{P_t G_t G_r \lambda^2 \sigma}{(4\pi)^3 K T_s B_n L_t L_r R_t^2 R_r^2} \tag{2.1}$$

where

S	= received power (in watts)
P_t	= power transmitted by the radar (in watts)
G_t	= power gain of the transmitting antenna
G_r	= power gain of the receiving antenna
λ	= wavelength of the signal frequency (in meters)
σ	= aircraft RCS (in square meters)
R_t	= distance from the transmitter to the aircraft (in meters)
R_r	= distance from the receiver to the aircraft (in meters)
N	= noise power at the input to the receiver (in watts)
K	= Boltzman's constant (1.38×10^{-23} joules/Kelvin)
T_s	= receive system noise temperature (in degrees Kelvin)
B_n	= noise bandwidth of the receiver (in Hertz)

The RCS σ is a characteristic of the aircraft dependent on the radar frequency, the shape of the aircraft, and the position of the aircraft relative to the radar transmitter and receiver. The RCS shape of an aircraft is complex and can vary by as much as 30 dB[5] for fractions of a degree change in aspect angle[6]. The amount of energy reflected (backscattered) to the receiver at a given angle can be represented from any angle as an isotropic sphere. The size of the sphere is not constant, but varies with the aspect angle. Also, the size of the sphere determines the amount of transmitted radar energy reflected back to the receiver as shown in Figure 2.4. The RCS σ of a target is different for monostatic radar than for bistatic radar. In monostatic radar, the radar receiver and transmitter are colocated and share the same antenna. For bistatic radar, the receiver and transmitter are separated and have different antennas. Bistatic is used as a countermeasure to low-observable technology to increase the detectability of "stealth" aircraft. A bistatic RCS σ_B is

[5]30 dB = factor of 1000; Decibel (dB) is a ratio of two numbers, say x/y, expressed with logarithms as dB = 10log(x/y).

[6]Aspect angle is the azimuth and elevation of the aircraft with respect to the radar transmitter and receiver.

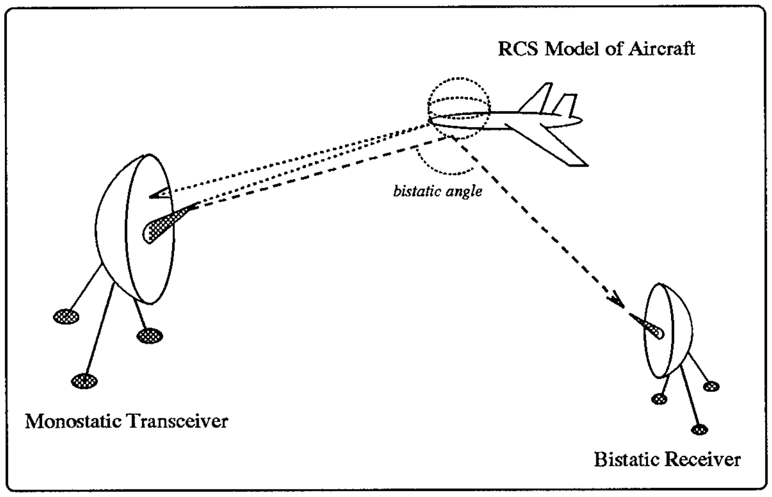

Figure 2.4 Monostatic and Bistatic Radar Model. The dotted line represents a monostatic radar signal, and the dashed line represents a bistatic signal.

more complex than a monostatic RCS because σ_B is a function of aspect angle **and** bistatic angle[7]. The distance of the aircraft away from the radar antennas and the size of its RCS determine the strength of the reflected radar signal. All these factors affect the probability of radar detection. The mission route planner tries to stay outside of the range of radar as much as possible, or uses mountains and valleys as terrain masking to "hide" the aircraft from the radar and prevent or limit detection.

Radar Cross Section Model. To adequately represent a three-dimensional RCS of a target in one degree increments would take $360^2 = 129600$ values for each radar frequency of interest. The RCS depends on the aircraft's orientation (i.e. azimuth, elevation, and degree of bank (or roll)), and the distances from the radar transmitter and receiver. The more aircraft orientations considered, the more complex the RCS pattern becomes. In addition, the monostatic RCS σ_M of a target is different than the bistatic RCS σ_B. There are various simplifications which can be made to reduce the RCS complexity. The most common is to treat the aircraft as a sphere so that the radar cross-section appears the same from any angle. The approach taken in (29, 50) used

[7]Bistatic angle is the angle formed by the intersection of rays from the transmitter and receiver to the target.

a single-valued sphere model for aircraft radar cross-section to simplify calculations. This model essentially treats the aircraft as a moving point and implies that the aircraft has the same radar image as seen from any direction at any angle. This simple model uses the same RCS value for all orientations of the aircraft (pitch, roll, yaw, azimuth, and elevation). In this current research, multiple RCS values are incorporated to represent front, side, top, bottom, and rear views of the aircraft. However, only monostatic radar is modelled, so the radar equation is simplified by the following relations: $G_t = G_r$, $L_t L_r = L_s$, and $R_t^2 R_r^2 = R^4$. Also, since radar systems are typically specified by their operating frequency, the following substitution can be made in Equation 2.1:

$$\lambda = \frac{c}{f} \tag{2.2}$$

where

c	= speed of light (3×10^8 meters per second),
f	= frequency (in Hertz).

Radar Probability of Detection. The probability of detection p_d is one criteria that the route planner tries to minimize. Detection depends on the strength of the reflected signal off the target. Therefore, radar detection and RCS are inter-related. Probability of detection p_d also includes statistical factors related to the signal-to-noise ratio (S/N), and probability p_f of a false alarm[8]. A target is detected when its return signal exceeds a specified threshold at the receiver. The simple model for this research does not include p_f. Only p_d is modelled[9] such that $S/N \geq 15$ implies $p_d = 1$, and that $S/N \leq 5.0$ means $p_d = 0$. Written in mathematical form:

$$p_d = \begin{cases} 1 & \text{if } S/N > 15, \\ \frac{(S/N)-5}{10} & \text{if } 5 < S/N \leq 15, \\ 0 & \text{if } S/N \leq 5. \end{cases} \tag{2.3}$$

[8]A false alarm is when the radar system identifies a target but none are present.

[9]The S/N differs for each radar, so in an actual implementation, the appropriate S/N values would be input as a characteristic for the specific radar.

This value can be calculated at every point along the path based on the radar system characteristics, aircraft RCS, and distance to the radar sites. A weight or cost is assigned to each point exposed to radar. With this model, the total exposure to radar along the route can be predicted based on the cumulative cost along the route. The total weight or cost of exposure to radar can then be minimized subject to the weighting of other route selection criteria.

2.3.2.2 Calculating Radar Detection: Static vs. Dynamic. There are two basic approaches to calculating radar exposure and probability of detection (59). Radar exposure is a boolean value of whether or not an aircraft is within line-of-sight of a radar site. Probability of detection quantifies this exposure. In the following discussion, "threat" is used to define radar exposure and probability of detection.

Static Threat Calculations. In the static approach, a grid is used to represent the terrain. The threat cost is calculated for each grid cell around each radar, and this threat matrix is "overlaid" on the terrain grid. The route planner then a path through the grid, searching for one with the lowest overall threat cost. Implementations of this technique are described in (48, 50, 93, 97, 109). An advantage to this approach is its simplicity. The threat matrix can be pre-calculated once the threats are known. Then, different routes can be selected using the same information, thus reducing duplication of threat calculations. There are several disadvantages:

1. A large amount of storage is needed to represent the threat and terrain information. In (50), the matrix has 250,000 points to represent 60 square miles. If threat and terrain values are represented by floating point numbers, at least 2 Mbytes of storage would be needed.
2. Threat calculations are based on a single value RCS. The static approach does not have the capability for using multiple RCS values.
3. Each time the actual radar threat changes, the entire threat matrix must be recalculated. Radar threats can change when radar sites are destroyed or jammed, or when tactical radar

is employed. Recalculating the threat matrix would take at least $O(kn)$ or longer, where k is the number of radar sites, and n is the total number of points in the 3-D grid.

Dynamic Threat Calculations. The dynamic approach calculates radar threat "on-the-fly". Using a 3-D grid, threat is calculated only as each grid point is encountered (29). The disadvantage to this approach is that calculations may be repeated if the same grid point is encountered multiple times. This may occur if the route planning algorithm uses a "backtracking" or "branch-and-bound" technique (24). However, there are several advantages:

1. No additional memory is required. Only the location and characteristics of the radar site are stored. Calculations are done as needed, and are not saved.
2. Calculations are only performed on potential route points. This can be a significant time savings over the static approach.
3. Updating the threat information involves only changing the information pertaining to the radar sites. Additions, deletions, and changes are updated quickly.
4. Multiple RCS values can be used in the threat calculations. The aspect angle of the aircraft can be used to determine the appropriate RCS value and calculate the corresponding threat cost. This is especially valuable for planning routes for low-observable aircraft to take advantage of the built-in "stealth" characteristics.

The advantages of the dynamic approach far outweigh the disadvantages in many applications. For the implementation of MRP in this research, the dynamic approach is the model this author has found which has the potential to adapt to multiple RCSs and a changing threat environment for a "real-time" route planner. Therefore, this is the model used for this research.

2.3.3 Aircraft Model. Two important features must be included in the aircraft dynamic model. One feature, RCS, is discussed in the preceding section. The other feature that must be modelled is the turn radius. A fighter aircraft has a smaller turning radius than a strategic bomber or tanker. And within the class of fighter aircraft, there is a wide variation of turning radius (11). The turning radius is a function of the speed of the aircraft, the "thickness" of the atmosphere (tighter turns are possible in thicker air close to sea-level), as well as the aircraft design. For example, an F-16 can reverse direction in 1600 meters at sea level at Mach[10] 0.85. An A-10 can complete the 180° turn in 800 meters flying at Mach 0.55 at sea level. A simplistic turn model in two dimensions allows turns of only 0 and ±45°. Thus, a simple grid path for a 180° turn would be four consecutive 45° turns as shown in Figure 2.5. To properly model this, the size of the grid cell

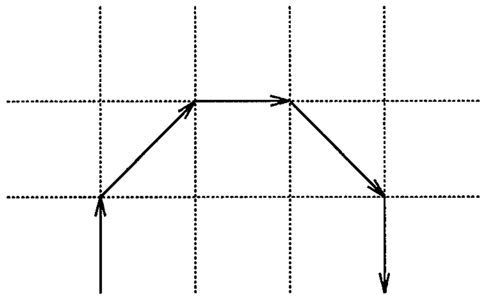

Figure 2.5 Model for 180° Aircraft Turn in 2-D Grid.

in the XY plane (latitude and longitude) must be large enough to allow the aircraft to complete its turn in the number of cells required to map the turn. This allows the algorithm to select a route that is within the aerodynamic capability of the aircraft. If a smaller grid cell is required to give the desired terrain resolution, then a more complex aircraft turn model may be needed which allows the aircraft to turn at angles between 0 and 45°. For three dimensions, angles of climb and descent must be modelled with and without turns. The model for aircraft movement used in this research is as shown in Figure 2.6. This model was chosen to reduce the number of points reachable from the current point. In this model, the aircraft can move forward in one of nine discrete directions along an axis or diagonal. Turns, climbs, and descents are limited to the constraints of the model. For this research, the model does not allow angles of descent or climb greater than 45° from horizontal.

[10] Mach 1 ≈ 760 miles/hour (1267 kilometers/hour).

Future research may explore the advantages and disadvantages of various improvements to this aircraft turn model.

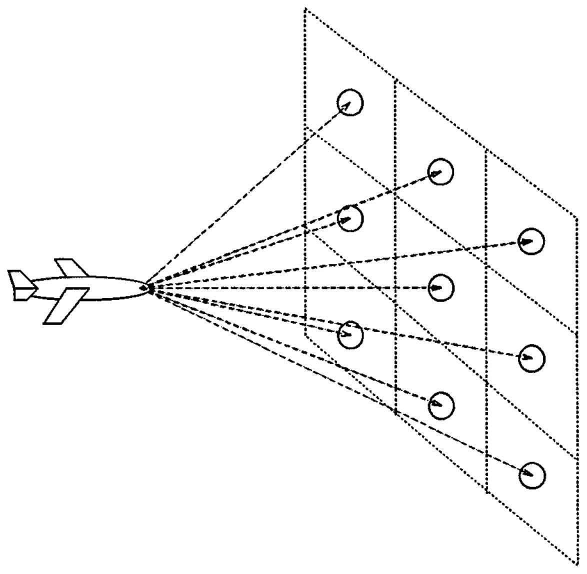

Figure 2.6 Simple Model for Aircraft Movement in 3-D Grid.

2.3.4 MRP Cost Function for Route Evaluation. Multiple factors must be considered when planning a real mission route. However, most of the factors can be grouped into two categories:

- Distance related factors, such as
  - Aircraft Range
  - Speed
  - Time
  - Fuel Consumption
- Threat avoidance factors, such as
  - Radar Exposure
  - Visual Detection
  - Communication Detection
  - Other Air Defense/Offense
  - Weather

To reduce the total number of criteria evaluated, this research evaluates only two criteria related to the categories — distance and radar exposure — for selecting an optimal route. In minimizing exposure to threat, it is important that the route planner consider not only the overall exposure, but the continuous exposure as well! For example, an aircraft may be more vulnerable to a continous 10-second radar exposure than to five separate 2-second exposures. The cost function that models these requirements follows a graph structure:

Given a set V of vertices, set E of edges, and a Graph $G = (V, E)$, let the length l_i of an edge e_i between two adjacent vertices $u, v \in V$ be defined by the function $l(u,v) = l(e_i) = l_i$, $0 < i \leq |E|$. Also, let a radar cost r_i for the same edge be defined by the function $r(u,v) = r(e_i) = r_i = w_i(p_d)_i l_i$, where w_i is the weighting factor of the radar cost, and $(p_d)_i$ is the probability of radar detection for e_i. Let the total cost of travelling along edge e_i be $c_i = l_i + r_i$. Then the total cost $C(s,g)$ of travelling along a path of N vertices from start s to goal g is simply the sum of the individual edge costs in the path as given by the following equation:

$$C(s,g) = \sum_{i=1}^{N-1} c_i = L(s,g) + R(s,g) \tag{2.4}$$

The length $l(u,v) = l_i$ is the Euclidean distance between u and v. The total length $L(s,g) = \sum l_i$, and the total radar cost $R(s,g) = \sum r_i$. The weighting factor increases with each consecutive edge exposed to radar according to the following equations:

$$w_i = \frac{(k+1)X}{k+X} \tag{2.5}$$

where

$$k = \begin{cases} 0 & \text{if } (p_d)_{i-1} \leq 0, \\ k+1 & \text{otherwise.} \end{cases} \tag{2.6}$$

Equation 2.5 increases the weighting factor w_i from an initial value of 1 to a maximum value of X as consecutive route points exposed to radar are encountered. This has the effect of increasing the radar cost for continuous radar exposure along the route. The factor X in the w_i numerator and denominator terms is a constant which is set at the beginning of the program and can be changed to give more or less penalty for continous radar exposure[11]. The value of the weighting factor affects the route selection by causing the route planner to search for longer unexposed paths before reconsidering the shorter exposed path. In other words, the route planner evaluates the optimality of all unexposed paths of length $l_j \leq l_i + r_i$ before reconsidering the exposed path l_i. Therefore, the solution with $C(s,g) = L(s,g) + R(s,g) = C_{bound}$ is the minimum cost path if for all paths from s to v, $C(s,v) \leq C_{bound} = C_{min}$. The number of paths that must be evaluated is

[11]Experiments have shown that $X = 4$ gives good results.

determined by how fast the C_{bound} is reduced to C_{min}, and how close all other path costs are to C_{bound}. The amount of radar coverage and the radar weighting factor determine the difficulty of this search problem. These factors are called "order parameters" (21) and they determine a "phase transition", or threshold below which the optimal solution can be found relatively quickly, and above which the optimal solution may take exponentially longer time to find due to the structure of the search space. The phase transition for MRP problems varies because the order parameters can change from one problem scenario to another. However, understanding the effect of the order parameters (radar cost factors) can give some insight into estimating the worst-case running time of a given MRP scenario.

2.3.5 0/1 Syntactical Model. The 0/1 Syntactical Model (24, 41, 91) is useful for modelling problems to be solved by search space techniques such as integer programming and the simplex method, or best-first search algorithms. A 0/1 Syntactical Model for the Bicriteria MRP Problem is developed here using the problem description from the previous section. In terms of MRP, to choose a route from point A to B, we first subdivide the path into many points between A and B. The distance between each point is the length of the edge $l(e)$, and we'll define the weight $w(e)$ as the probability of being detected by radar when travelling along the edge connecting the two nodes[12]. The MRP problem can then be mapped to a 0/1 syntactical model by considering each path either selected (marked with 0), or not selected (marked with 1). By forming this into an $n \times n$ boolean matrix of 0's and 1's, we have our syntactical model. Therefore, the 0/1 syntax for the problem is:

$$\text{Minimize} \quad z = \sum_{i=1}^{m} l(e_i)\xi_i \tag{2.7}$$

[12]This is a simplification of the problem to only two criteria. For additional criteria, we can combine all criteria related to distance, such as fuel consumption and time of flight, into the $l(e)$ function; all vulnerability criteria, such as radar detection, weather, and night missions, can be combined into the $w(e)$ function.

$$\text{subject to} \quad \sum_{i=1}^{m} w(e_i)\xi_i \leq W_{max} \tag{2.8}$$

$$\xi_i = 0 \text{ or } 1 \tag{2.9}$$

$$m = |E| \tag{2.10}$$

where $\xi_i = 1$ (or 0) depending on whether e_i is (or is not) in the path from A to B; W_{max} is the maximum allowable weight (i.e. total probability of detection along the path); and m is the total number of edges in E.

This model can be used with the cost function described in Section 2.3.4 to give a formal notation for the MRP problem. This notation is used in the next section as part of a discussion of comparing MRP to a Shortest Path problem.

2.4 Can Mission Routing Problem Simplify to Shortest Path Problem?

The Single-source Shortest Path Problem (SSP) is a single criteria path search problem that seeks to minimize the sum of the arc weights between vertices in the path from start to goal. It can be solved in $O(n^2)$ time using Dijkstra's algorithm (a "greedy" search technique), where n is the total number of vertices to be searched (14). The Mission Routing Problem (MRP) is a multi-criteria path search problem in which the objective is to find the best path that optimizes each of the individual criteria. In order to map MRP to SSP, the multiple criteria must be combined into a single cost function assigned to the distance (arc) between path points (vertices). For the case of two criteria (distance travelled and radar threat), the optimal solution would lie on a line in the plane (see Figure 2.7). The slope of the line is determined by the linear combination of the two criteria, such that

$$C = L + W \tag{2.11}$$

where

C	= Total Cost of the path,
L	= Total Length of the path, and
W	= Total Weight or Radar exposure cost of the path.

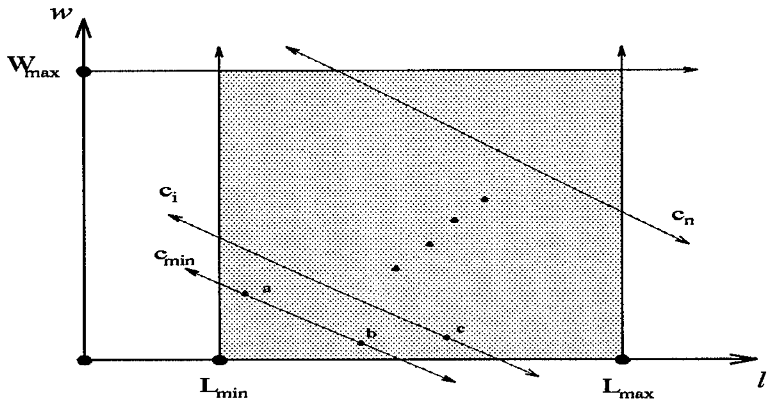

Figure 2.7 Graph Showing Solution Cost Region for Length and Weight. Length is the total path length between two points. Weight is the sum of the weights of the individual nodes in the path.

L_{min} represents the shortest straight line distance from the Start to Goal. W_{max} represents the radar exposure cost of the L_{min} path if it had maximum radar exposure at all points in the path. $L_{max} = L_{min} + W_{max}$. The shaded area in the figure represents the possible path solutions where $C \leq C_{max} = L_{max}$. To minimize the cost C, any solution which lies on the line for minimal cost c_{min} would be considered "optimal". Therefore, points a and b would have equal cost, and would be chosen over c, which has greater cost. Note that a route with higher radar cost and shorter distance can have the same total cost as a longer route with lower radar cost. Each would be an "optimal" solution in terms of the objective function and the weights assigned to each criteria. All the c_i lines have the same slope determined by the selection of the weights. Selection of weights is important in determining the slope of the cost c_i lines, especially when the number of criteria increases. For example, with a linear combination of three criteria, the solution becomes a line in a volume. With four or more criteria, it becomes even more complicated and the challenge lies in developing the appropriate weighting to assign to each criteria to produce an "optimal" solution. Here, optimality is defined as the line with smallest valued intercepts along each axis of Figure 2.7.

Problems With the SSP Approach. SSP algorithms assume that the shortest path between points in a digraph is made up of smaller shortest subpaths. This optimal substructure

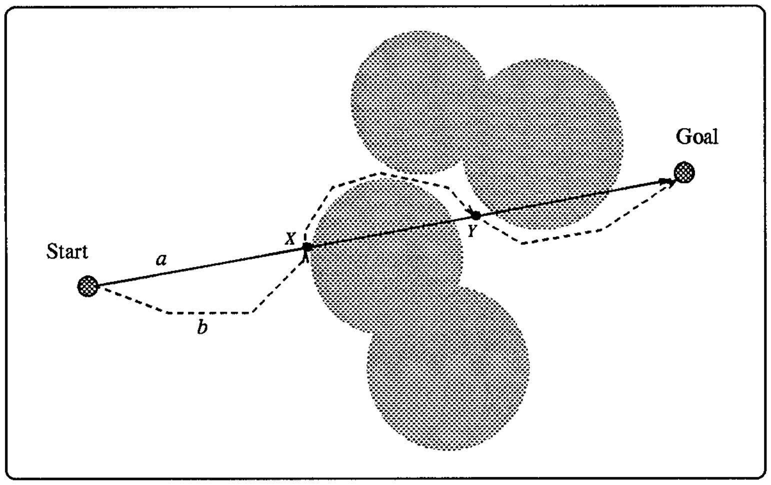

Figure 2.8 Graph showing how optimal substructure for shortest path problem does not always apply for MRP problem. Path a has shorter length to X, but path b has lower overall cost.

is required by both dynamic programming and the greedy method to find the shortest path in polynomial time (26). However, MRP does not always have this optimal substructure. An example of this is shown in Figure 2.8. Because of the physical constraints on aircraft movement, an aircraft on path a cannot make the turn at X, so it must move forward and get exposed to radar. This is because outgoing paths from a point are determined by the direction of the incoming path to the point. An aircraft on path b can avoid the radar entirely, so that it has a lower overall cost than a. However, shortest path (SP) algorithms choose a over b because path costs are evaluated only from the last point encountered. To apply SP algorithms to the MRP problem, the MRP graph must be transformed to an SP graph such that each edge in MRP is mapped to a vertex in SP. An example of this is shown in Figure 2.9. For the aircraft model used in this research (see Section 2.3.3), there are 24 directions (edges) into and leaving a point. If each of these edges must be mapped to an $l \times n \times m$ three-dimensional matrix of points, there are approximately $24lnm$ directed edges. With this transformed graph, an SP algorithm such as Dijkstra's algorithm can be applied to the problem. As stated above, Dijkstra's algorithm takes $O(N^2)$ time to find a solution, where $N = 24lnm$. Does this help? Not much. For example, the matrix used in (50) and (29) was

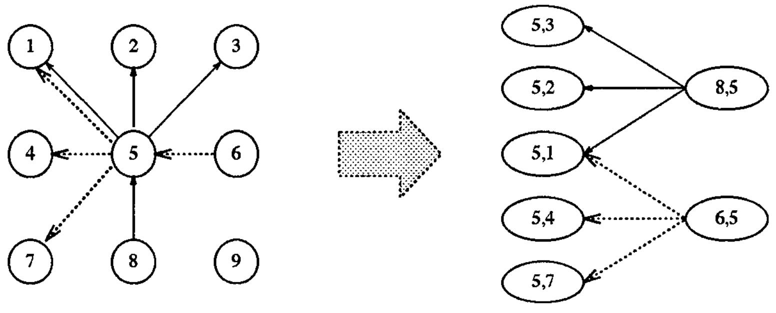

Figure 2.9 Example Transformation of MRP Graph to Shortest Path Graph. Each of the edges in the MRP graph is mapped to a vertex in the Shortest Path graph.

$100 \times 100 \times 25$. Transforming this matrix and running Dijkstra's algorithm would take on the order of 3.6×10^{13} elementary operations[13]. To accomplish this in one hour or less would take a computer system that can perform the elementary operation in less than 1.8×10^{-10} seconds. This requires better than gigaflop performance that can only be achieved by massively parallel computers. For example, the Paragon at Sandia National Laboratories used 1,840 nodes to achieve a benchmark of 102 Gflops (120).

Another drawback of this approach is calculation of distances and radar costs. The transformed graph makes it more difficult to calculate straight-line distances required for determining radar costs. This adds complexity to the problem. The amount of complexity added cannot easily be determined without a detailed analysis, which may be another topic for future research.

A third disadvantage is that Dijkstra's algorithm requires an adjacency matrix for the vertices. In the transformed graph, each vertex can be adjacent to at most nine other vertices, based on the turn model used for the aircraft. Therefore, for the example above, the size of the adjacency matrix must be $\geq 9 \times 24 \times 100^2 \times 25 = 51.5$ Mbytes. This does not include the memory that may be required for other data structures, as well as the computer operating system and the executable program.

[13]An elementary operation is the computer operation(s) needed to evaluate each n_i, $1 \leq i \leq N$, whose time can be bounded above by a constant depending only on the particular implementation used (machine, programming language, etc.) (14:9).

To summarize, Shortest Path (SP) algorithms can be used to solve the MRP problem if the appropriate transformations are made. However, the time and space requirements must be addressed in order to make an SP approach practical for a real-time MRP system. Heuristic methods (such as the A* algorithm) may be employed to reduce the overall time and space complexity of the problem.

2.5 Complexity of General MRP Problem

This section assumes that the reader has a thorough understanding of the theory of NP-Completeness. An introduction to the theory of NP-Completeness could take several pages in itself, so for more information, the reader is referred to (40, 77) and (54:151-183). A complexity analysis is used to determine the worst case running time, and the worst case memory requirements for a given problem. This section addresses the time complexity of the MRP problem, and the space complexity is addressed in the section on implementation. The Mission Routing Problem falls into the general class of multicriteria path search problems. Selecting a path that optimizes multiple criteria can be a difficult task. In many cases, multicriteria path problems take exponential time to find "optimal" paths (17, 16, 28, 98). A class of problems which have known exponential worst-case running times is called NP-Complete[14].

2.5.1 Comparison to Other NP-Complete Problems. MRP is similar to the orienteering problem (45, 46, 57, 69, 100), the selective travelling salesman problem (78), the weighted region problem (88, 86, 89), robot motion planning (17, 16, 65, 68, 94, 98, 119), the obstacle avoidance problem (7, 18, 23, 56, 60, 94), and other constrained path problems (19, 22, 25, 58, 83, 91, 103, 113, 117). All these problems can be categorized as NP-Complete. They are all related because every NP-Complete problem can be polynomially transformed to every other NP-Complete problem. However, some problems are more closely related to MRP than others. For example, if multiple

[14]Most of this section is derived from Garey and Johnson (40).

targets are considered where the sequence of attack must be optimized, the MRP problem becomes a subset of the Travelling Salesman Problem, which is NP-Complete. Even if the problem is simplified to one aircraft against one target and only two route criteria, it remains NP-Complete. This last case is referred to as the Bicriteria MRP problem, and it is proved to be NP-Complete in the next section using a transformation from the PARTITION problem.

2.5.2 Proof of NP-Completeness. By some definitions, NP-Complete applies only to decision problems (40). A decision problem Π is one that can be answered with a "Yes" or "No." An optimization problem can be formulated as a series of decision problems where each decision problem sets a lower bound to the criteria until the solution that satisfies the lowest bound is found. In that sense, the MRP optimization is a series of MRP decision problems, and the theory of NP-Completeness can be used to analyze the time complexity of the MRP problem. Formally, a language L is defined to be NP-Complete[15] if $L \in NP$ and, for all other languages $L' \in NP$, $L' \propto L$. Proving NP-completeness for a decision problem Π takes four steps (40:45):

1. show that $\Pi \in NP$,
2. select a known NP-complete problem Π',
3. construct a transformation f from Π' to Π, and
4. prove that f is a polynomial transformation.

These steps are followed to show that the bicriteria MRP problem is NP-Complete. The Shortest Weight-Constrained Path (SWCP) Problem is said to be NP-Complete by a transformation from the PARTITION problem (40, 98). A similar argument is given in (119). The following analysis shows the transformation from PARTITION to SWCP, and finally to the Bicriteria MRP (BMRP), using the notation from (40).

Definition 2.5.1 PARTITION
INSTANCE: Finite set A and a size $s(a) \in Z^+$ for each $a \in A$.
QUESTION: Is there a subset $A' \subset A$ such that $\sum_{a \in A'} s(a) = \sum_{a \in (A-A')} s(a)$?

[15]The term "complete" comes from the recursion theoretic notion of a language being "complete" for a class with respect to a given type of reducibility if it belongs to the class and every member reduces to it.

Lemma 2.5.1 *PARTITION is NP-complete.*

Details of this proof are given in (40:60-62). Next, PARTITION is transformed to SWCP.

Definition 2.5.2 SHORTEST WEIGHT-CONSTRAINED PATH (SWCP)

INSTANCE: Graph $G = (V, E)$, length $l(e) \in Z^+$, and weight $w(e) \in Z^+$ for each $e \in E$, specified vertices $s, t \in V$, positive integers K, W.

QUESTION: Is there a simple path in G from s to t with total weight W or less and total length L or less?

Lemma 2.5.2 *SWCP is NP-complete.*

Proof[16]: SWCP $\in$ NP, since a nondeterministic algorithm need only guess a simple path $P(s,t)$ in G and check in polynomial time if the sum of the edge lengths is less than L and the sum of the edge weights is less than W. Next, transform PARTITION to SWCP. Consider a graph $G = (V, E)$ with $2n$ edges $e \in E$ and $n+1$ vertices $v \in V$ such that $t = v_1$ and $s = v_{n+1}$. Let the set $A = E$ and $a = e$. Draw two edges joining each v_i to v_{i+1}, a "top" edge with length 0 and weight a_i, and a "bottom" edge with length a_i and weight 0. (See Figure 2.10.) From PARTITION, let $s(a) = w(e_i^{bottom}) = a_i$ for $1 \leq i \leq n$, and $s(a) = l(e_k^{top}) = a_i$ for $k = n + i$; $1 \leq i \leq n$. It is clear that this transformation can be done in polynomial time by $2n$ simple assignment statements assigning the values a to e. To finish the proof, let $L = \sum_{k=n+1}^{2n} l(e_k^{top})$, and $W = \sum_{i=1}^{n} l(e_i^{bottom})$. It is clear that $L = W = \sum i = 1^n a_i$. Let $A' = E' = \{e_i^{bottom}\}$. This gives $E - E' = \{e_i^{top}\}$. A path of length $\leq L$ and weight $leqW$ yields a partition into a set of top edges, and a set of bottom edges. Therefore, PARTITION $\propto$ SWCP. $\square$

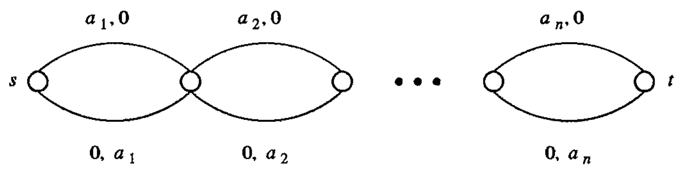

Figure 2.10 Reduction from Partition.

SWCP is also NP-Complete for directed graphs, the above proof also holds if a constant c_i is added to the lengths (weights) of the top and bottom edges. It is only important that the difference between the top and bottom edge lengths (weights) be equal to a_i.

Using the two previous lemmas, BMRP can now be shown to be NP-Complete:

Definition 2.5.3 BICRITERIA MISSION ROUTE PLANNING (BMRP)

INSTANCE: Graph $G = (V, E)$, length $l(e) \in Z^+$, and radar cost $r(e) \in Z^+$ for each $e \in E$, specified vertices $s, t \in V$, positive integers L, R.

QUESTION: Is there a simple path in G from s to t with total radar cost R or less and total length L or less?

[16]Proof of SWCP adapted from (98).

Theorem 2.5.1 *BMRP is NP-complete.*
Proof: Restrict to SWCP by allowing only instances for which $r(e) = w(e)$, $l(e_i) - l(e_{n+i}) = a_i$, and $r(e_i) - r(e_{n+i}) = a_i$. $\square$

These bicriteria problems are said to "weak" NP-Complete (40), which means that for restricted inputs, pseudo-polynomial time algorithms may exist for them. Knowing the effect on complexity of restricting the set of inputs for a problem is important because, in many applications, the inputs that actually occur may have special properties (like symmetry) that allow a polynomial-bounded solution (5:333). In fact, a pseudo-polynomial time dynamic programming solution is given for the PARTITION problem with restrictions on the input size (40:90-95). When more than two criteria are included in the MRP problem, the problem becomes NP-Complete in the "strong" sense (40) (*i.e.* it becomes a much harder problem than the bicriteria case). Proof of this would involve transformation of the 3-PARTITION problem for the case of three criteria, 4-PARTITION problem for the case of four criteria, etc.. The MRP problem becomes increasingly harder to solve when more route selection criteria are included in the problem. And formulating the criteria into a single measure of performance may lead to solutions which are far removed from the solution the route planner actually would have preferred (10). Analyzing the complexity of more than two criteria for the MRP problem is a topic for future research. In the next section, the specific time complexity of the Bicriteria MRP Problem is discussed.

2.5.3 Time Complexity of Bicriteria MRP Problem. NP-Complete problems have time complexity $O(c^n)$ worst case. This complexity analysis attempts to identify the variable parameters that contribute to this worst case. This analysis assumes that the multiple criteria for evaluating the path are combined into a single additive cost function that is bounded above by a constant[17], C_{max}. It is also assumed that there are no negative values for the cost function (*i.e.* no negative arc weights). For this analysis, only two criteria are considered, distance and radar threat. For each pair of points, the distance is the length $l(e)$ of the edge connecting adjacent points in the path.

[17]If multiple cost functions are used, the lower and upper bounds to the complexity become much worse.

The radar threat is the probability of detection $0 \leq p_d \leq$ times a radar multiplication factor k times the distance $l(e)$. The maximum cost per edge is therefore $l(e) + kl(e) = (k+1)l(e)$. Consider a function $c(i,j) = (k+1)l(e_{ij})$ which determines the arc cost of moving from point i to point j. For any arbitrary arc, this function evaluates to values in the range $C_{min} < c(i,j) \leq C_{max}$. C_{min} is obtained by setting all criteria to their lowest bounds for a unit travel (that is, going from one point to any adjacent point), and multiplying by the number of points L visited in the most direct route from point i to point j. The most direct route is defined as the shortest distance obtained by summing the lengths of line segments connecting consecutive points in the path. The direct route may be longer than the straight-line Euclidean distance due to digitization bias discussed in Section 2.3.1.4 and shown in Figure 2.3. C_{max} is obtained by setting all criteria to their upper bound for a unit travel and multiplying by L. The cost of the optimal route from i to j is therefore bounded above by the maximum cost of a direct path between those points. In other words, the time complexity of finding an optimal solution does not depend on the total number of points in the search space. Instead, the search time depends on the maximum length for an optimum path. Given that the multiple criteria are combined into a single cost function, the maximum length can be determined a priori by first defining the following:

- a constant $k = \frac{C_{max}}{C_{min}}$ representing the ratio of the cost of worst-case travel (fully exposed to radar) to the cost of optimum travel (with no radar exposure), and
- L representing the length of a "direct" path. L may not be unique due to digitization bias.

The longest path L_{max} with cost $\leq C_{max}$ is one with no radar exposure. The longest path with no radar exposure must be equal to kL. Consequently, the length of the optimal path is bounded above by $kL = L_{max}$. The paths of length $\leq L_{max}$ that have endpoints at start S and goal G are confined to the area of an ellipse drawn such that the sum of the radii is $\leq L_{max}$ as shown in Figure 2.11. Thus, only the paths within the shaded area of the ellipse must be evaluated to determine the optimal solution to the cost function $c(S,G)$. Therefore, the total number of points N that must be considered in a search for an optimal solution is the area of an ellipse $O((kL)^2)$ for two dimensions, and the volume of an ellipsoid $O((kL)^3)$for three dimensions.

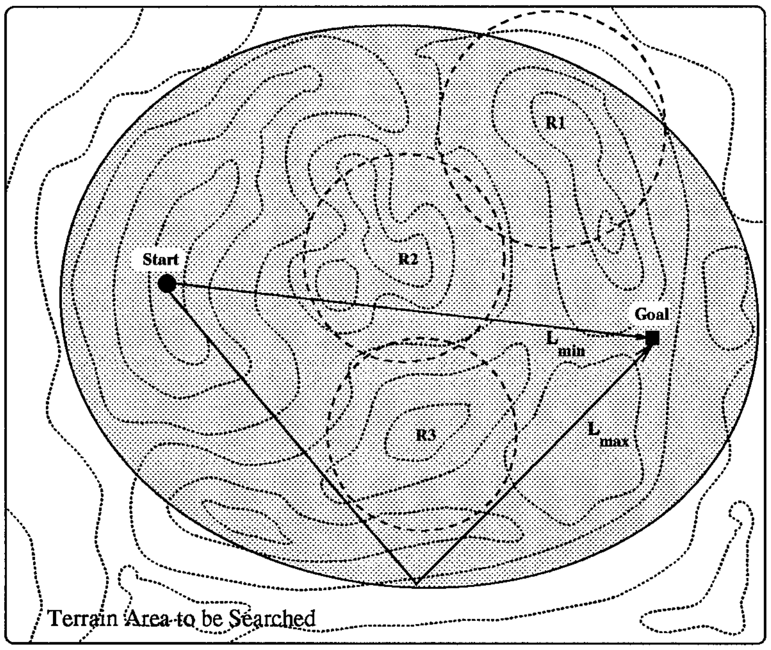

Figure 2.11 Example of MRP Terrain Area to be Searched. The shaded area represents an ellipse with foci at Start and Goal, and L_{max} = sum of the radii.

The number of paths that can be generated from N points is determined by the branching factor b defined as the constant number of successors (or next points) that can be reached from each point in N (92:92). From each point, b new paths can be generated. Thus, there are at most b^l paths of length l. The optimal path L^* contains the number of points in the range $L \leq L^* \leq kL$. Therefore, the maximum number of paths that must be generated is determined by the branching factor b, and the maximum number of points in the optimal path as follows:

$$\sum_{i=L}^{kL} b^i = (b^{kL} - b^L)b/(b-1) \tag{2.12}$$

The lower bound for the number of paths generated is readily determined if we assume that at each point, the successor is chosen such that it lies on the optimal path. Thus, MRP has lower bound $\Omega(bL)$. Therefore, the bounds on MRP time complexity are:

$$\Omega(bL) \leq MRP \leq O(b^{kL}) \tag{2.13}$$

Typical values for b and k are 9 and 4 respectively, with $L \cong 100$ or larger. The branching factor b is determined by the aircraft flight model, and represents the number of next points reachable by the current point as shown in Figure 2.6. The variable k is the radar multiplication factor that is set according to the level of threat assigned to radar. The higher the value for k, the more the route planner tries to avoid radar in the route selection. With these typical values for b and k, even the largest, fastest computers in the world would not be much help in the worst case. Fortunately, time complexity is closer to the lower bound in practice. Thus, parallel processing may offer solutions to complex MRP problems in times fast enough to meet mission needs. Tables 2.1 and 2.2 summarize the need for efficient algorithms to solve the MRP problem by showing that even a hundredfold speedup in processing time only increases the input handling capability of an exponential algorithm by a small amount. Therefore, the challenge is to combine a two-fold approach of developing techniques to reduce the size of problem instances while at the same time producing efficient implementations that can provide speedup over current methods.

Time	Maximum problem size		
complexity	1 sec	1 min	1 hour
n	1000	6×10^4	3.6×10^6
n^2	31	244	1897
n^3	10	39	153
2^n	9	15	21

Table 2.1 Limits on Problem Size as Determined by Growth Rate (3).

One approach to an efficient implementation is to transform the MRP problem to a Shortest Path (SP) problem in order to satisfy the constraints for an optimal path substructure, as discussed in Section 2.4. By performing the transformation to an SP problem, the complexity of the problem

Time complexity	Maximum problem size before speedup	Maximum problem size after speedup
n	s_1	$100s_1$
n^2	s_3	$10s_3$
n^3	s_4	$4.64s_4$
2^n	s_5	$s_5 + 6.64$

Table 2.2 Effect of Hundredfold Speedup.

is reduced. The SP problem has complexity $O(N^2)$ to search an area of N points. However, as shown in Figure 2.11, only $(kL)^3 \leq N$ points must be searched to find an optimal path. Therefore, by satisfying the constraint for an optimal path substructure, the upper bound for MRP is reduced to $O((kL)^5)$. Substituting this into Equation 2.13 produces the following relation:

$$\Omega(bL) \leq MRP \leq O(k^5L^5) \leq O(N^2) \tag{2.14}$$

Applying this approach to the MRP reduces the problem complexity such that a real-time application for larger (more realistic) scenarios may be possible.

2.6 Summary

This chapter presents a detailed discussion of the Mission Route Planning (MRP) problem, describes the models used in this research, and shows that the general MRP problem is NP-Complete. To the extent that these goals were met, this chapter forms the basis for this research, and the building block for future research. The MRP objective was to automate the route planning process to optimize route selection based on at least two criteria of distance travelled and exposure to radar. The choice of models is driven by the problem requirements. The radar avoidance requirement is to utilize multiple RCS values without significant time or space penalty. This radar requirement drove the selection of a standard grid model for terrain. All other model choices are based on efficiently and effectively implementing the terrain model while meeting the other problem requirements. The radar cost model exploits the structure of the grid terrain model in order to implement a weighted radar cost scaled to the number of consecutive path points exposed to radar. The complexity analysis is based on the assumption of a grid representation for the terrain. In the discussion of Shortest Path (SP) Algorithms, it was argued that for certain restrictions and mappings, the MRP problem could be solved using SP algorithms. However, the mappings may increase the number of points to be searched, which may affect the overall execution time. The importance of the SP discussion is that it gave the conditions for mapping the MRP problem to the SP problem. An SP approach to MRP reduces the time complexity upper bound from exponential to polynomial time.

The remainder of the thesis discusses methods to refine the discrete computational models in this chapter, and presents the design, experimentation and results analysis of implementations using a parallel best-first algorithm (A*).

III. Mission Route Planning (MRP) Methods

3.1 Introduction

The previous chapter described the MRP problem and the various models employed for manipulation by computers. This chapter presents the different automation methods that may be used to manipulate the models to obtain solutions to the MRP problem. Automated methods for mission route-planning (MRP) requires a multi-discipline approach involving techniques from operations research, electrical engineering, computer engineering, and computer science. MRP is one of many path finding problems. Much research has been conducted to find effective and efficient algorithms to solve general path finding problems, such as the traveling salesman problem (38, 42, 78, 79, 90, 95), and orienteering (45, 46, 57, 69, 100). Other path problems include robot motion planning as described in (17, 16, 36, 56, 65, 84, 89, 94, 102, 117, 119), and path planning for autonomous underwater vehicles (AUVs) in three dimensions as reported in (18, 60, 76, 116). Research for air vehicle mission routing is described in (48, 82, 108, 109) for helicopters, and fixed-wing aircraft routing is described in (6, 12, 13, 29, 33, 50, 67, 64, 71, 80, 93, 97). However, research pertaining to solving MRP in three dimensions applied to fixed-wing aircraft with multiple radar cross-sections (RCSs) and dynamic radar calculations has not been published. A simpler modelling approach performs dynamic radar calculations using a single RCS value (29, 31). Other research on aircraft routing typically pre-process radar calculations before performing the path search algorithm (48, 50, 51, 82, 97, 108, 109). Different techniques have been used to automate the general MRP process with varying degrees of success. This chapter specifically reviews the literature referenced above on the application of computers to solve general MRP problems.

3.2 Goal of MRP Methods

The goal of a particular MRP method affects the selection of the algorithm or solution technique. Is the goal to find an "optimal" solution, or just a "good" feasible solution? To find a feasible solution, a problem is mathematically formulated into one with n inputs, and a subset of the inputs is obtained that satisfies some criteria. Any subset that satisfies these criteria is called a *feasible* solution (14, 74). A feasible solution that either maximizes or minimizes a function (called the *objective function*) defined on the problem is an optimal solution. There is normally an obvious way to determine a feasible solution but not necessarily an optimal solution. In most cases, feasible solutions can be obtained in polynomial time[1]. Most of the MRP systems developed for operational use produce only feasible solutions based on user input. However, finding an optimal solution from the set of feasible solutions for MRP is much more difficult, as discussed in the previous chapter (Section 2.5). "Deterministic search" algorithms can be used to find optimal solutions. To find a good feasible solution, not necessarily optimal, "stochastic search" algorithms may provide better performance than deterministic algorithms (4, 14, 44). A list of techniques under each category is provided below:

- Search Strategies to solve MRP
  - Deterministic Methods
    1. Breadth First
    2. Depth First
    3. Best First
    4. Dynamic Programming
  - Stochastic Methods
    1. Potential Fields.
    2. Genetic Algorithms
    3. Simulated Annealing.
    4. Monte Carlo and Las Vegas.

These concepts are also discussed in the following sections:

3.3 Mission Route Planning Description

Military planners recognize the need to reduce the time it takes to plan a mission. For example, generation of an Air Tasking Order (ATO) typically takes 72 hours if done manually

[1]For MRP, a feasible solution is any path with endpoints at the start and goal. A depth-first algorithm (8, 24, 26) can find a feasible solution in polynomial time $O(n^2)$.

System	User	Acronym
AAFMPS	U.S. Air Force	Advanced Air Force Mission Planning System
AFMSS II	U.S. Air Force	Air Force Mission Support System II
AMPA	U.K. RAF	Advanced Mission Planning Aid
AMPS	U.S. Army	Airborne Mission Planning System
CAMPAL	Netherlands RNLAF	Computer Aided Mission Planning at Airbase Level
CHAMPS	U.K. RAF	Chinook and Hercules Advanced Mission Planning System
CINNA 3	French Air Force	
CIRCE 2000	French Air Force	
DART II	U.S. Air Force	Desktop Auto-Routing Tool II
IMOM	U.S. Air Force	Improved Many on Many
MARPLES	Italian Air Force	Military Aircraft Route Planning Expert System
MOMS	U.S. Marines	Map, Operator and Maintenance Station
NAMPS	U.S. Marines	Night-attack AV-8B Mission Planning System
PA	U.S. Air Force	Pilot's Associate
STAMPS	U.S. Air Force	Strategic/Tactical Automated Mission Planning System
TAMPS	U.S. Navy	Tactical Aircraft Mission Planning System
TEAMS	U.S. Navy	Tactical EA-6B Mission Support

Table 3.1 Sample of Mission Planning Systems Developed by NATO Countries [from (1, 2, 57)].

(57). The ATO is then used by operational units to plan their respective parts of the mission. Planning the flight route requires information on the aircraft, weapons, terrain, target location, threats, and weather (6, 52, 57). When done manually, selection of a route relies on the experience and judgment of the planner. Rarely is there time to optimize this route. Therefore, research has been conducted to determine how best to automate this process (29, 50, 66, 97, 109). Many computer systems have been developed as aids to mission planning, but few have automated route-planning functions. Most current computer systems developed for MRP require manual entry of the route points, and the computer then evaluates the "survivability" of the selected route (52, 59). Among NATO[2] countries, dozens of mission planning systems have been developed. A small sample is outlined in Table 3.1 [from (1, 2, 57)]. Of those systems listed in the table, only DART II, AFMSS II, and PA provide a capability for automatic generation of an "optimal" route. Optimality is defined differently for each system. DART II is an analysis tool for testing new concepts in low-observable technology. It uses "dynamic programming" as the route-planning

[2]NATO = North Atlantic Treaty Organization, a peacekeeping body for defending Europe, comprised of European countries, United States, and Canada.

algorithm (67). PA was designed to be a real-time "co-pilot" for single-seat fighter aircraft that provides, among other tasks, real-time automatic route generation. PA was a DARPA[3] initiative that never reached operational deployment. PA used the A* algorithm implemented in hardware with the goal of providing real-time speed for optimal route-planning (27). AFMSS II is currently under development. The automatic route-planning component of AFMSS II is called CLOAR — Counter Low-Observable Auto-Router. According to the requirements document (34), CLOAR will be designed to plan optimum routes for multiple aircraft (max 32), against multiple targets (max 18), with up to 10,000 threats, in less than eight hours. This author is skeptical that these requirements can be met for an optimal route-planning system due to the complexity of the problem (see Section 2.5). The algorithm design for CLOAR is still under development.

3.4 Deterministic Search Methods

Deterministic methods are those that have predictable, repeatable search behavior with a given set of inputs. This is in contrast with stochastic methods where the search behavior can change with a given input based on the random probabilities of the method used. Deterministic search methods can yield optimal solutions for an objective function, whereas stochastic search methods may or may not give optimal answers (14, 93, 109).

3.4.1 Uninformed Search. The simplest of deterministic search algorithms is uninformed search. A search strategy is uninformed if the location of the goal does not affect the number of input points explored, except for the termination conditions (96:36). The most efficient variations of uninformed search algorithms are bounded depth-first search with backtracking, and breadth-first search with uniform cost[4]. A brief discussion is given here, in order to compare them to more efficient algorithms.

[3]DARPA = Defense Advanced Research Projects Agency.
[4]For a more detailed discussion, consult any good text on algorithm design such as (3, 5, 14, 26, 96).

In depth-first search (14:171), the depth of the path is extended by one each time. If a dead-end is reached, or the depth-bound is exceeded, backtrack to the last "good" point. A feasible solution is obtained when the goal is reached. For an optimal solution, all paths with costs less than or equal to the current best path must be explored. This can potentially explore all paths in the search space.

In simple breadth-first search (14:182), the search expands in a uniform front away from the start node. First, all nodes one step away are expanded, then all nodes two away, etc. until a solution is found, or all nodes have been explored. With uniform cost, the search front is based on equal (*i.e.* uniform) cost from the starting point, rather than the uniform number of points in the path. With uniform cost approach, an optimal solution is obtained when the goal is reached. This method may also cause the search to explore all possible paths in the search space.

3.4.2 Dynamic Programming. This technique is a recursive formulation of breadth-first search which, for spaces with regular structures, may yield analytical expressions for the optimal cost or optimal policy (14, 75). The problem is subdivided into smaller and smaller subproblems which are solved one-by-one and assembled until the complete problem is solved. A list is kept of the generated subproblems so as to make sure that the same problem is never solved twice. The optimal solution is arrived at by employing the principle (property) of optimality, which requires that for any intermediate state of the problem and the respective intermediate solution for this state, the solutions of the subsequent subproblems must constitute an optimal solution sequence with regard to the problem state resulting from the stated intermediate solution (5, 14).

Dynamic programming has been used to solve shortest-path problems, and has been applied to solve the MRP problem. IBM used dynamic programming to develop a route-planner for the Rotorcraft Pilot's Associate program (109). The planner was designed for helicopters, and was implemented on massively parallel processors (CM-2 machine). The sequential time complexity for the algorithm was reported as $O(n^3)$. The paper described a two-dimensional experiment in which the planner selected the best route over a $100 \times 100 = 10{,}000$ square mile grid in 5.9 seconds using 10,000 processors. However, computing in three dimensions would greatly increase the execution time of the program. For example, adding discrete elevation increments of 0.25 miles from ground level to a height of 3 miles would increase the number of grid cells by a factor of 12, from 10,000 to 120,000. Since the algorithm is $O(n^3)$, a naive estimate for the increased execution time of this 3-D example would be $12^3 \times 5.9$ seconds $= 2.8$ hours. This does not include added overhead caused by increased interprocessor communication. Also, the threat areas are modelled statically using cellular automata. This approach may be suitable for helicopters which have a large RCS, but it cannot provide optimal routes for low-observable aircraft.

Reported results indicate that dynamic programming performs better than heuristic[5] search (the A* algorithm) to plan optimal paths for an autonomous underwater vehicle (60). Dynamic programming required less memory and executed four times faster than heuristic search. The algorithms were designed to run on a sequential machine. However, general conclusions cannot easily be drawn from the comparisons based only on experimental results. The performance of both algorithms is greatly dependent upon the objective function for each one. Dynamic programming evaluates paths to all points in a search graph, and its efficiency relies on the optimal substructure of the shortest path problem (as discussed in Section 2.4), and the fact that it does not repeat calculations. The A* algorithm will explore fewer paths if its heuristic objective function is monotonic[6]. Dynamic programming with a good objective function may outperform the A* algorithm with a poor heuristic function. Details on the objective functions were not given, so the conclusions drawn from the paper cannot be applied in the general case.

In (87), a dynamic programming algorithm is described which computes the shortest path through polygonal obstacles in two dimensions. No experimental results are given for the algorithm, but the theory of development is sound, and the algorithm is mathematically proven to be efficient

[5]Heuristics are criteria, methods, or principles for deciding which among several alternatives promises to be the most effective to achieve some goal (96:3).

[6]More information on the A* algorithm is provided in Section 3.6.

in computational time and memory usage. However, extending this to three dimensions may make the problem intractable. For example, it has been reported that multicriteria path problems for three-dimensional robot motion planning is NP-Complete (16, 17, 28).

In (121), Yen develops a new strategy for partitioning the search grid among parallel processors to increase the efficiency of his dynamic programming algorithm. This implementation was only 2-D, and the results were not as good as 2-D experiments for the IBM route-planner (109).

3.4.3 Informed Best-First Search: A.* Informed best-first search algorithms use heuristics. For use in computation, a heuristic is mapped to a data structure or function that a selected algorithm can manipulate to select the current global best point from among all points evaluated thus far. The performance of these heuristic algorithms is directly dependent upon the formulation of the heuristic evaluation function $f(n)$. When $f(n)$ is additive, and the problem is formulated using an "OR" graph, the best-first algorithm is called A*. This is the most popular of best-first algorithms because when certain conditions for the algorithm are met, A* is said to dominate (*i.e.* is more efficient than) all other algorithms that have access to the same heuristic (96:85). The A* algorithm is discussed in more detail in Section 3.6. For a thorough discussion of other best-first techniques, the reader is referred to (96).

3.5 Stochastic Methods and Probabilistic Algorithms

Stochastic methods use probabilities to search for feasible solutions. The method can be translated as follows: it is sometimes preferable to choose the next step at random, rather than to spend time determining which alternative is best (14). Algorithms that employ stochastic methods are called probabilistic algorithms. Due to the random nature of alternative selection, a given method can have varying solutions and varying run times that are not necessarily optimal. However, if the cost of finding an optimal solution is prohibitive, probabilistic algorithms may provide good solutions in much faster time. A good introduction to probabilistic algorithms is given in (14).

Five methods discussed here are Potential Fields, Genetic Algorithms, Simulated Annealing, Monte Carlo and Las Vegas techniques.

3.5.1 Potential Fields. Potential fields is another technique that does not fall into the standard categories of deterministic search algorithms discussed previously. Path-planning algorithms based on potential fields borrow from the concepts of electromagnetism to establish differences of potential between data objects. Opposites attract, and likes repel. The goal point is given a strong positive potential, and the starting point (source) is given a potential less than (*i.e.* more negative than) the goal. The moving point is assigned the same potential as the source. Obstacles are assigned different potentials with values less than the source. The moving point is than repelled by the source and obstacles, and attracted towards the goal. The primary disadvantage with this method is that it tends to get trapped in local field minima and may never reach the goal (116). Also, it cannot guarantee an optimal solution.

3.5.2 Genetic Algorithms. One of the most promising of the stochastic search methods is the field of genetic algorithms. Genetic Algorithms (GAs) apply genetics and the theory of evolution to computer problem solving (44, 85). Search strings are encoded like a gene structure such that threat areas are given "poorer" genes, and safer regions of the map are given "better" genes. For MRP, a gene string represents a path from point A to point B. Using a population of many simultaneous gene strings, a path gradually evolves over time in such a way that the "fittest" tend to survive (93, 97, 109, 111). By definition, a low-cost path is more fit to survive than a high-cost path. With proper formulation of the problem, the longer the algorithm runs, the greater the probability of finding an optimal solution. An advantage of GAs is that they are easily parallelized. However, GAs do not necessarily find the "best" solution if one exists. The search sometimes converges to a poor solution even when a better solution is close by. In addition, dynamic radar calculations with multiple radar cross-sections cannot easily be modelled with genetic algorithms.

Olsan (31, 93) took the work of Grimm (50, 51) and developed a parallel genetic algorithm implementation to compare to Grimm's parallel A* implementation. Like Grimm's work, the radar exposure was statically calculated and represented by a 3-D threat matrix overlaid onto the 3-D terrain matrix. In Olsan's GA implementation, the solutions obtained were at least 20% worse than Grimm's A* approach, and took longer to complete.

Pellazar (97) developed a genetic algorithm (GA) for MRP that was remarkably similar to Olsan's (93) work. He compared his GA implementation to a dynamic programming (DP) approach and found that the dynamic programming approach performed faster and produced better solutions, on average, than his GA approach. However, his experiments showed isolated cases where the GA found an optimal solution faster than the DP algorithm. These results cannot be generalized because his experiments were in two-dimensions, not three, and the examples he described were simple enough that a person could find an optimal path quickly without the aid of a computer.

Stiles (109:177-178) experimented with developing a route-planner based on GAs. His results were unsatisfactory. The routes selected by the GA implementation were complex, less than optimal, and required longer run-times than deterministic methods. This is partly because GAs are relatively new, and not understood completely. There is no clear understanding on how to select the GA parameters to produce the most effective and efficient search implementations.

Thangiah (111) developed a GA approach to a two-dimensional path planner. He modelled a single aircraft, a single target, and a hostile environment. The aircraft movement was encoded to one of eight directions (north, northeast, east, southeast, etc.). Radar was statically modelled as threat cells or points. Reported results were no better than the GA references listed previously.

3.5.3 Simulated Annealing. Simulated Annealing (SA) algorithms are based on annealing processes in thermodynamics (15, 70). Annealing is the heating and slow cooling of an object. Simulated on a graph data structure such as in MRP, the algorithm would "heat" all points within a specified distance from the starting point. As a minimum, the distance must be greater or equal to the straight-line distance to the target. Then over successive iterations, the points would be cooled according to a probabilistic cooling schedule that defines the rate of cooling. "Threat" areas such as radar are assigned higher energy states so that the algorithm attempts to avoid those areas during the cooling process. Over time the algorithm will settle on a minimum temperature which should give an optimal path. However, optimality is greatly dependent on the cooling schedule, and the algorithm has a tendency to get trapped in local minima, although some claim that given enough time SA will find the optimal answer (70).

3.5.4 Monte Carlo and Las Vegas. Monte Carlo algorithms are those that are always fast and probably correct, whereas Las Vegas algorithms are probably fast and always correct (54:316). A Monte Carlo algorithm occasionally makes an error, but it finds a correct solution with high probability whatever the instance considered. However, the number of errors and times of occurences are unpredictable. This is unacceptable for MRP. A Las Vegas algorithm, on the other hand, always produces a correct answer given enough time. However, due to the randomness of the selection decisions, the algorithm may require vastly different execution times for the same problem instance (14). This is also unacceptable for the MRP problem. This author has not found any published reports of applying Monte Carlo or Las Vegas algorithms to the MRP Problem.

3.6 A Search Algorithm Description*

The A* algorithm is an informed best-first graph search which uses an additive cost heuristic to quickly home into the best solution. The A* algorithm was developed in 1968 by Hart, Nilsson and Raphael (55), although many specific computational search algorithms employed additive cost functions prior to this generalization. A* has gained wide acceptance because with proper choice of the heuristic, A* dominates (*i.e.* performs better) than all other algorithms with access to the same heuristic. The heuristic is part of the evaluation function $f(n)$ that assigns a cost of moving along a given path in the graph. The 'A' in A* means that it uses an additive cost function. The

'*' refers to optimality in that if certain conditions on $f(n)$ are met, A* will always find a solution that optimizes the cost function $f(n)$. To apply A* algorithms to 3-D path-planning problems, the physical regions of the 3-D space are discretized to form a graph of discrete points or nodes, and the cost function $f(n)$ is evaluated from node to node. The cost function $f(n)$ at a node is the sum of the actual cost g from the start node to the current search node, plus the cost h from the current node to the goal node, as shown in Figure 3.1 and described in the following equation (101:75) and (92, 96):

$$f = g + h \tag{3.1}$$

where

g	= actual cost from starting node to current node,
h	= cost from the current node to the goal, and
f	= total cost of the path from the starting node to goal.

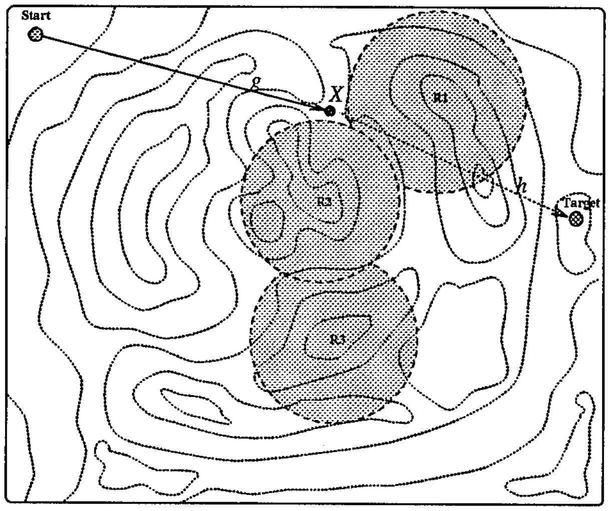

Figure 3.1 Evaluation function for A* is $f(X) = g(X) + h(X)$ for any point X.

When the goal is reached, $h = 0$, and $f = g$. The objective of the search process is to find an optimal path, *i.e.* one that optimizes f. Perfect knowledge of h would guarantee that the search would always be on the optimal path. On the other hand, letting $h = 0$ turns A* into a breadth-first algorithm. In most problems, h is not known explicitly, but may be estimated to fall within certain bounds. This estimated value for h is referred to as the heuristic function. The next section discusses the importance of the heuristic function and presents the pseudocode description of the A* algorithm.

3.6.1 Pseudocode for the A Algorithm .* The A* algorithm contains two lists, OPEN and CLOSED, and operates on a graph G of N nodes. OPEN contains successor nodes that have not been evaluated. CLOSED contains all nodes that have been evaluated. A successor of node n, labeled n', is a node that is adjacent to n. Let $c(n, n')$ be a cost function that evaluates the cost of moving from node n to its successor n'. Based on these descriptions, the pseudocode for the A* algorithm is as follows (96:64):

```
Algorithm A*
  1. Put the start node $s$ on OPEN.
  2. If OPEN is empty, exit with failure.
  3. Remove from OPEN and place on CLOSED a node $n$ for which $f$ is minimum.
  4. If $n$ is a goal node, exit successfully with the solution obtained by tracing back the
     pointers from $n$ to $s$
  5. Otherwise expand $n$, generating all its successors, and attach to them pointers
     back to $n$. For every successor $n'$ of $n$:
     (a) If $n'$ is not already on OPEN or CLOSED, estimate $h(n')$ (an estimate of the
         cost of the best path from $n'$ to a goal node), and calculate $f(n') = g(n')+h(n')$
         where $g(n') = g(n) + c(n, n')$ and $g(s) = 0$.
     (b) If $n'$ is already on OPEN or CLOSED, direct its pointers along the path
         yielding the lowest $g(n')$.
     (c) If $n'$ was found on CLOSED and required pointer adjustment, reopen it.
  6. Go to step 2.          □
```

The optimality and efficiency of any A* implementation[7] depends upon the heuristic function $h(n')$. For optimality, the heuristic must be *admissible* (96:77). The heuristic is admissable if it is less than or equal to the actual (optimal) value h^*.

Definition 3.6.1 *A heuristic function $h(n)$ is said to be* admissible *if $h(n) \leq h^*(n), \quad \forall n \in 1..N$.*

[7]This description implies that the steps are executed sequentially. Additional steps must be added for a parallel implementation of the A* algorithm. These are discussed in the next chapter.

An efficient A* algorithm is one that explores as few nodes as possible. The most efficient algorithm would be one with $h(n) = h^*(n)$. Since most problems cannot be formulated such that $h(n)$ matches $h^*(n)$ exactly, another condition *monotonicity* must be applied to obtain efficiency. Monotonicity is not required for the A* algorithm to find the optimal solution, but it is important for reducing the number of node expansions performed (96:82).

Definition 3.6.2 *A heuristic function $h(n)$ is said to be* monotonic *if it satisfies:*
$h(n) \leq c(n, n') + h(n'), \quad \forall n, n' |\; n'$ *is a successor of* n.

If $h(n)$ is monotonic, the algorithm never reopens CLOSED nodes, and therefore an explicit CLOSED list is not required. This can improve the efficiency of sequential implementations of the A* algorithm. In addition, with a monotonic heuristic the optimal solution is obtained as soon as the goal is reached.

3.6.2 Variations of A.* Variations of A* have been proposed to overcome the memory requirement for the standard A* algorithm. Although A* is optimal and efficient with a monotonic heuristic, optimality is guaranteed only if all generated paths are stored on the OPEN list until they are evaluated. If interim paths are generated faster than they are evaluated, the OPEN list can grow quickly in size, and may require more memory than is available[8]. However, modified A* algorithms, such as in (72, 73, 81, 117), reduce the memory usage of the standard A* algorithm. Most of these modified versions are variations of Iterative Deepening A* (IDA*) (72, 81). This is more of a depth-first approach with backtracking (75:305). Only the current path is stored so that the total required memory is reduced. IDA* expands the "best" paths to a certain depth (or length). "Best" is determined by the heuristic. If the Target is not found at the current depth, the depth is "iteratively deepened" until the Target is found. Some claim that IDA* performs as fast or faster than A* without the large memory requirement of A* (72:106). However, IDA* may perform redundant work across iterations, and this redundancy may cause IDA* to have longer execution

[8]This was observed in this research when less than four processors were used. The parallel algorithm developed in this research reduced the growth of the OPEN list when number of processors ≥ 8. Details given in Chapter 6.

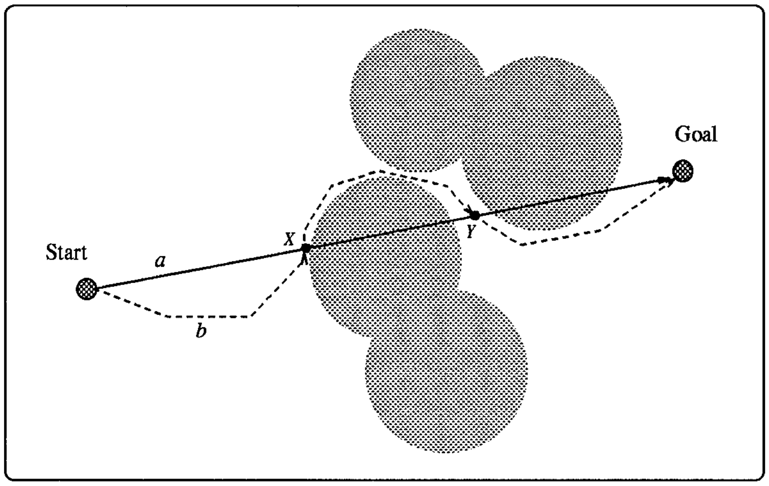

Figure 3.2 Situation Violating Monotonicity.

times than the standard A* approach. This is problem dependent and is greatly affected by the heuristic [9]. This author believes that IDA* cannot perform better than A* for MRP because of the redundant work that would be performed. Verifying and validating this claim is a subject of future research.

3.6.3 Multicriteria Search and Heuristic Estimates. Formulating multiobjective (multicriteria) problems to fit into A* search algorithms is discussed in (107). Developing a monotonic heuristic for a multicriteria A* search algorithm may be difficult. Each of the separate criteria may exhibit monotonicity, but when they are combined into one cost function, the resulting function may not be monotonic. An example of this is shown in Figure 3.2 for the MRP problem with two criteria: distance and radar exposure. The criteria are separately monotonic, but their combination is not monotonic. The heuristic $h(Y)$ along path a is greater than $h(\text{Goal})$ along path b, thus violating monotonicity. The sum of the criteria does not increase monotonically because one criterion, radar exposure, is greater than the other criterion, distance, along some paths, but less than the distance along other paths. For the combination to be monotonic, one criterion must be strictly greater than (or less than) the other criterion. That is not the case in this example, and in fact,

[9]It cannot be overstated that the performance of any A* approach is greatly dependent on the heuristic function $h(n)$.

this combination is not admissable because h(Goal) along path a is greater than along path b. In mathematical terms, $h_a(\text{Goal}) > h_b{}^*(\text{Goal})$, which violates admissability. A monotonic heuristic for this problem would therefore be limited to estimating only one criterion.

3.6.3.1 Information in the Heuristic. If h is admissable, an optimal solution can be obtained with A*. The closer h is to h^*, the more informed h is (96:81). A fully informed heuristic would be $h = h^*$. Lack of any information would be $h = 0$ which is a breadth-first search. Consequently, for admissability, $0 \leq h \leq h^*$. Those heuristics h which are close to zero cause A* to explore many unproductive nodes not on the optimal path, and require a large amount of unproductive nodes to be stored on the OPEN list. An h close to h^* focuses the A* search towards those nodes on paths which are optimal or very close to optimal, and reduces the number of nodes stored on the OPEN list. This analysis implies that a more informed heuristic would lead to more efficient algorithm execution. In practice, a more informed heuristic may take longer to calculate than one that is less informed! Therefore, in choosing the best heuristic, the algorithm designer must tradeoff the benefits of a more informed h against the time required to calculate h. A simple heuristic that can be calculated quickly may obtain solutions faster but with more required storage than a more informed and complex h that requires more time to calculate but with less overall memory required. For example, Grimm's MRP heuristic combined radar cost and distance by performing a depth-first search for a fixed distance towards the target, calculating distance and radar cost along the way (50). The remainder of the cost was estimated using straight-line distance to the Target. This approach put more information in the heuristic, but took time to calculate and did not guarantee monotonicity. Droddy's heuristic estimated distance only (29). Although it had less information, it was monotonic, faster to calculate, and generally performed better than Grimm's heuristic.

3.6.4 Parallel A Algorithms.* The execution speed of serial processors is approaching the physical limits of semiconductor devices. In addition, many real-world problems are too large to be handled by a single serial computer due to time and memory constraints. Multiple processors running in parallel may be used to further decrease program run times. This applies to the A* algorithm and the MRP problem, as well. In general, there are three major considerations involved with the design of a parallel A* algorithm (75).

- Termination condition of the algorithm.
- Implementation of the OPEN list (distributed versus centralized)
- Implementation of the CLOSED list (explicit versus implicit)

Termination condition of the algorithm. In the sequential version of the algorithm, the optimal solution is obtained when the goal is reached because at any time, the current best solution is at the top of the OPEN list. For parallel A*, there are P processors expanding points from the OPEN list, and each may be searching a different area of the search space. When one processor reaches the goal, there are $P-1$ other processors still searching that may lead to better solutions. Therefore, the parallel algorithm must delay termination among the P processors until the global best solution is found. Termination occurs when processors have emptied their OPEN lists.

Implementation of the OPEN list (distributed versus centralized). Each processor must access the OPEN list for each expansion of a point along the path. Therefore, the OPEN list must be accessible to all processors. A centralized (global) OPEN list allows each processor access to the global best next point to expand, but may create a communication bottleneck, especially in message passing architectures. According to (32, 49), a global OPEN list has limited scalability, and is best suited for shared-memory systems[10]. Distributing the OPEN list among the processors would reduce the communication overhead, but processors would be expanding only the local best which may not be the global best next point. In addition, local distributed OPEN lists

[10]This author agrees that a global OPEN list has limited scalability because contention for the OPEN list multiplies the communication overhead as the number of processors is increased. This has been validated through experimentation reported in Chapter 6.

have additional problems with consistency among *Open* lists, starvation from uneven work distribution, and duplicate work among the processors (32). A hybrid approach was used in (29, 50) with the best information from all processors managed by a Control Process, and the remaining information kept in local OPEN lists on each worker process.

The best approach to implementing the OPEN list is problem dependent! Most of the published literature stresses the importance of a global OPEN list to keep the A* algorithm focused on the global best heuristic estimates. This is a valid concern for single criterion problems, but may not be appropriate for multicriteria search problems. The heuristic may not capture information for all the criteria. If so, a global approach using this heuristic is an inefficient guide of the search which forces the algorithm to explore unproductive areas of the search graph. In fact, for bicriteria MRP, this research discovered that searching for the best estimates in a certain direction or subarea of the search graph (*i.e.* a local best approach) performs significantly better than searching for the global best among all the processors. Better performance is achieved by distributing the OPEN list among processors and reducing the interprocessor communication. This is explained in Section 3.6.4.1, and validated by experiments[11].

Implementation of the CLOSED list (explicit versus implicit). In the sequential algorithm, a monotonic admissable heuristic function may preclude the need for an explicit CLOSED list. Thus, a point on OPEN that has been expanded does not need to be explicitly stored on a CLOSED list because it will never have a lower heuristic value due to the monotonicity of the heuristic function. However, in a parallel implementation, each processor is expanding a different area of the search space. Unless the processors share information, their search paths may overlap, resulting in some duplication of work. Because of this, an explicit CLOSED list is usually required to reduce duplication and redundant searches between processors. For example, an explicit CLOSED list was not used in (29, 50). Experiments showed that for some MRP scenarios this

[11]See Chapter 6 for more details.

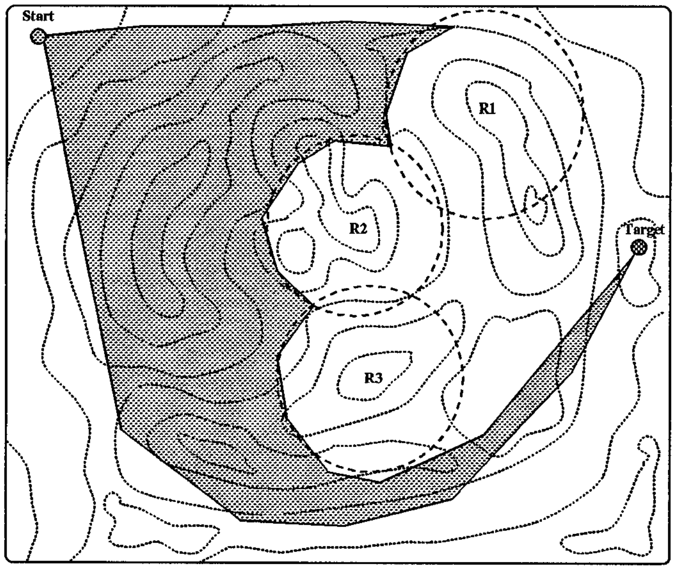

Figure 3.3 Sequential version of A* search. Shaded area shows the area explored by A* with a heuristic based only on distance.

resulted in hundreds of duplicate nodes produced and shared among the processors. Consequently, efficiency was greatly reduced and execution times were unacceptable for a real-time application. The details of the experiments are given in Chapter 6.

3.6.4.1 Best Parallel A Approach.* In the serial A* version for MRP, the search front is composed of paths with equal total costs as shown in Figure 3.3. It is inefficient to maintain this front globally in a parallel approach. This inefficiency was demonstrated in (29, 50) and resulted in slow execution times. A better approach is to search for the best path in a specific direction. Each processor is assigned to a specific direction and is guided towards the goal along that specific direction. This results in a more autonomous A* search as recommended by (32, 49, 115). An example of this is shown in Figure 3.4 for four directions handled by four processors. A comparison of Figures 3.3 and 3.4 reveals that the parallel search method may explore fewer nodes than in the serial A* search. If implemented correctly, this may provide a very efficient and fast parallel A* algorithm. This is the approach taken in this research investigation.

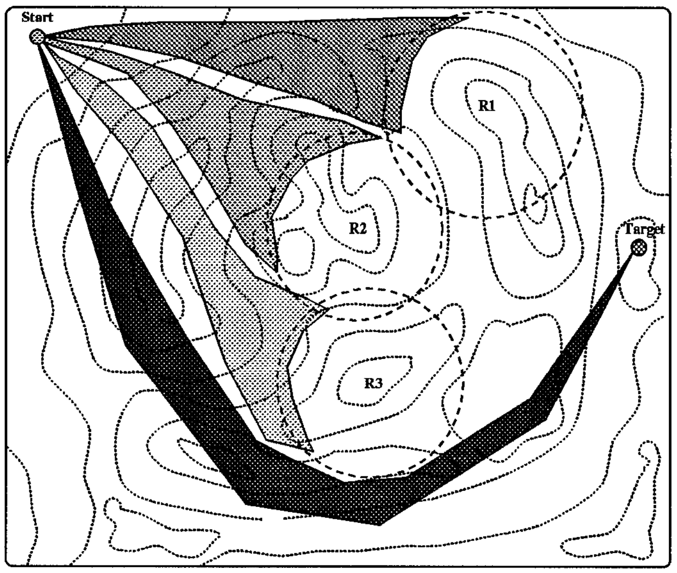

Figure 3.4 Parallel Version of A* search with 4 Node Processors. Goal is reached faster with potentially less unproductive area explored.

3.6.5 A Path-Planning Implementations.* In general, the A* algorithm is the most reported and most used algorithm for route-planning problems. The popularity of A* is supported by its good performance and applicability over a wide range of route-planning problems. A small selection of those references that are most closely related to this research is discussed next.

Grimm (50) developed an optimal route-planner based on the A* algorithm implemented on parallel processors for increased performance over conventional serial processors. An optimal route was generated for one aircraft against one target. Two criteria were used to plan the route — distance travelled and radar detection. Terrain was simulated by a three-dimensional mesh representation. The aircraft was modelled as a point mass with movement in one of 24 directions. The radar threat was modelled statically as a matrix overlaid on the terrain model. Up to five radar threats were represented in the threat matrix. The "cost" associated with each route was a weighted combination of the distance travelled and the radar probability of detection. The optimal route was selected as the route with the lowest overall cost. Grimm's approach did not work consistently because a non-monotonic heuristic was used. The heuristic was a combination of both distance and radar threat, and could overestimate the actual cost of moving from a given node to the target. It "looked ahead" a specified distance and then estimated the rest of the way to the goal.

Droddy (29) continued the work begun by Grimm. The significant change he made to Grimm's work was the elimination of the static radar threat matrix and associated high memory storage cost. To make this change, Droddy modelled the aircraft as a spherical Radar Cross Section (RCS) and calculated radar probability of detection as needed. In this way, both bistatic and monostatic radar threats could be modelled. Droddy's code had faster execution times than Grimm, but Droddy's lack of a *closed* list for the A* algorithm made the code unreliable[12] for many types of problems where the target is surrounded by radar threats. Elimination of the *Closed* list is valid for sequential algorithms if the heuristic function is monotonic. The heuristic used by Droddy was simply the straight-line distance from the current node to the target. Although this heuristic is monotonic, it is a gross underestimator of the actual cost when radar and terrain are in the path. In the parallel version of A* implemented by Droddy, this heuristic and lack of a *Closed* list caused the program to generate unproductive and duplicate nodes on the *Open* list. This, in turn, decreased the efficiency of the algorithm and caused erratic performance for most test cases.

In (18), A* is used to plan paths in three dimensions for an autonomous underwater vehicle (AUV). Details of the algorithm and the heuristic function are not given. The algorithm is implemented on a serial processor. A quadtree structure is used to reduce the number of search nodes. However, the penalty for this is that a suboptimal path may result because the path is constrained to go through the centers of quadrants (18:80). The experiments showed that a small change in the number of path constraints resulted in a large change in execution time. This validates the need for parallel processors to handle large multicriteria 3-D path problems such as MRP.

In (94), Pal develops an A* approach to robot path planning in three dimensions. However, it can't really be called an A* approach because it seeks a "good" solution rather than an "optimal"

[12]Here, unreliable means that the code did not execute properly, i.e. did not terminate, generated errors or yielded incorrect solutions.

solution. By sacrificing optimality, this method is able to provide collision avoidance solutions quickly for movement of a robot arm. Reported execution times were less than a minute to run on a PC/AT-386 for most experiments. However, the physical search area was constrained to a small fixed size. In addition, a simple polygon model was used for the search area rather than a standard grid. These design decisions reportedly worked well for the robot arm implementation but do not apply, in general, to MRP.

In (110), Teng incorporates time constraints, as well as distance and threat criteria, in his modified A* algorithm for robot path planning in two dimensions. This work is quite similar to the IBM work of Stiles reported in (109). Both used a CM-2 massively parallel hypercube network and reported path planning experiments in two dimensions. However, Teng used a much larger grid size (512×512) than Stiles ((100×100), and had more path constraints. Reported results showed that Teng's parallel A* approach performed faster than Stiles's parallel dynamic programming approach. This lends support to our choice of the A* algorithm over dynamic programming for application to the 3-D aircraft route-planning problem.

3.7 Summary

Mission Route Planning is a complex and time-consuming task that requires automation. The various algorithmic approaches to automate MRP can be categorized as follows:

- Deterministic Search
  - Bounded Depth-First Search with Backtracking
  - Breadth-First Search with Uniform Cost
  - Dynamic Programming
  - Heuristic Search (A*)
- Stochastic (Probabilistic) Search
  - Potential Field Methods
  - Genetic Algorithms
  - Simulated Annealing.
  - Monte Carlo and Las Vegas.

Only deterministic search can guarantee optimal solutions. A summary of the important characteristics of each of the Deterministic Search approaches is provided in Table 3.2. Of these

Parallel Search Algorithms			
ALGORITHM	MEMORY	TIME	COMMENTS
Depth First	Requires little memory, Best feature.	Varies greatly depending on where in search graph a solution is located. Can require prohibitive amount of time.	Branch and bound, and backtracking can greatly reduce time.
Breadth First	Can require prohibitive amounts of memory.	Varies greatly, depending on where in search graph a solution is located.	Branch and bound, and backtracking can greatly reduce time and memory required.
Best First	Can require prohibitive amounts of memory. Uses less than Breadth First.	Solutions are consistently quickest, but can be longer than Depth First depending on cost bound.	Characteristics depend on the variation used (AO*, A*, Z*, etc.).
Dynamic Programming	Same as Breadth First	Depends on Objective Function. May be comparable to A*	Recursive Breadth First approach. Efficient because operations on same input are stored, not repeated.
A*	Same as Best First.	Same as Best First	If heuristic is admissable, and monotonic, A* dominates all other algorithms with access to the same heuristic.
IDA*	Memory the same as Depth First.	Claimed to be the same or better than A* (72).	Still undecided issues on relative speed, especially in parallel version.

Table 3.2 Memory and Time Comparisons of Parallel Search Algorithms (42:2-22).

approaches, dynamic programming and heuristic search are the most efficient because they reduce work by other methods and expand the fewer paths. In this research for optimizing MRP, A* was chosen because, if a monotonic heuristic is used, then A* out-performs any other algorithm which also finds the optimal solution (96:85). In the next chapter, we translate this general discussion of the A* algorithm into a high-level design.

IV. Requirements and High-Level Design

4.1 Introduction

This chapter presents the requirements and the high-level design of the MRP problem. The development of a MRP software system can be grouped into four phases: requirements, design, implementation, and testing (105:122). The requirements phase includes analysis and specification of the problem. The design phase can be further divided into high-level design and low-level design. Through each phase, software engineering principles (39, 114) are employed to provide a methodical approach to the development. The development process begins with the highest levels of abstraction, and detail is added through successive iterations (*i.e.* refinements) until the final implementation is completed. Abstraction is the selective examination of certain aspects of a problem which are important in characterizing the problem to a given level of detail (104). For Mission Route Planning (MRP), the abstractions are the models used to represent the physical world (terrain, aircraft, radar, ...) which capture the crucial aspects necessary for finding an optimal mission route. The highest level of abstraction is in the requirements specification.

In this chapter, the MRP problem is first specified in general terms independent of the specific terrain model employed. From this general specification, an English word description of an algorithm design is presented, followed by a formal design using UNITY[1] notation. Using the logic and theorems of UNITY, proof of the correctness of the design is given. This proof is valid regardless of the terrain representation chosen. In the next design iteration (refinement), the specific terrain representation is selected and described using object-oriented design techniques. Additional refinements concerning the heuristic, and the abstract data type for the OPEN list are presented to complete the description of the high-level design. The high-level design begins with the requirements specification of the problem.

[1]UNITY is an acronym for Unbounded Nondeterministic Iterative Transformations (20:8).

4.2 Requirements

The specification of the problem begins by defining the requirements and then generating the design models needed to meet the requirements. In this research, the MRP problem is simplified to finding the least cost flight path for a single aircraft from a starting point to a single goal (target). The time complexity of this problem is described in Sections 2.4 and 2.5. This research defined the formal requirements of this MRP problem as follows:

> Given a starting point S, a goal G, a set of all flight paths P from S to G, and a cost function $C(p_i)$ that evaluates the cost of each $p_i \in P$, find the least cost path from S to G.
>
> $$\exists\, Best = p_i \in P \quad \text{such that} \quad \forall p_j \in P,\ C(p_i) \leq C(p_j). \tag{4.1}$$

The objective of this research is to find *Best* in less than a specified time bound (4.5 minutes for the bicriteria MRP problem in this research). Parallel computers are selected as an efficient means to meeting this real-time objective.

4.2.1 Models.

The first physical model that must be specified is the representation of the earth. At this level of abstraction, it is modelled simply as a graph of vertices and edges[2]. The model of the earth is the union of the terrain and sky (flight space of the aircraft).

$$\begin{aligned}
V &= \{Vertices\} && (4.2)\\
E &= \{Edges\} && (4.3)\\
Earth &= Graph(V, E) && (4.4)\\
Terrain &\subset Earth && (4.5)\\
Sky &= Earth - Terrain && (4.6)\\
\text{Path } p_i &= [S, v_1, v_2, \ldots, v_k, G],\ v_k \in V,\ k = (|p_i| - 2) && (4.7)
\end{aligned}$$

The definition of $Vertices$ does not need to be specified at this level of abstraction. In addition, models for radar and the aircraft are not required to specify the problem description at this level. The model of the terrain is purposely kept general at this stage so that the formal

[2]An alternative to a graph structure is a tree structure. However, in a tree, nodes at the same level cannot be adjacent to each other (*i.e.* no cycles allowed). This is not representative of the physical terrain model.

high-level design is not dependent on the type of terrain model selected. Those refinements are made later in this chapter (see Section 4.4), after the proof of correctness of the general design.

4.3 High-Level Design

The primary purpose of the high-level design is to provide a template for selecting the underlying algorithm used to manipulate the computer models in order to obtain the optimal path. The heart of the design is the algorithm or search process, and the rest of the implementation is built around it. As determined in Chapter 2, under certain conditions Shortest Path algorithms (in (20), and (75)) can be used to find solutions to MRP, but they necessarily require searching through all vertices[3]. The A* algorithm (see (96:64)) is a "better" approach[4] and was selected as the search algorithm for finding the optimal path because, with proper choice of heuristic, it dominates (is more efficient than) all other algorithms with access to the same heuristic (96:85). Recall from Chapter 3 that the cost function for A* is

$$f(S,G) = g(v_i) + h(v_i) \tag{4.8}$$

where

$$\begin{aligned} g(S,v_i) &= \text{actual cost from starting vertex } S \text{ to current vertex } v_i, \\ h(v_i,G) &= \text{cost from } v_i \text{ to the goal } G, \text{ and} \\ f(v_i) &= \text{total estimated cost of the path from } S \text{ to } v_i \text{ to } G. \end{aligned}$$

The strength of A* lies in proper choice of the heuristic function $h(v_i)$. When $h(v_i) = 0$, the A* algorithm reduces to a Shortest Path algorithm which relies only on local information to search for the optimal path (96). If $h(v_i, G)$ is a good estimate of the remaining path cost from v_i to G, the A* algorithm can greatly reduce the number of vertices it must evaluate to find the optimal path from start S to goal G. To correlate back to the requirements, the A* cost function is equivalent to the path cost function:

$$f(v_i) = C(p_i), \quad \text{if } v_i = G \tag{4.9}$$

[3]Dynamic programming is sometimes used as the algorithm for Shortest Path problems (20:446).

[4]If the heuristic h is set to 0, A* is equivalent to Dijkstra's algorithm (75:265). The heuristic reduces the number of points searched, which makes A* a "better" algorithm.

The next step is to develop a pseudocode and formal notation to describe the operation of the A* algorithm for MRP.

4.3.1 Pseudocode. The pseudocode is an English word outline of the steps of the algorithm. To tailor the A* algorithm to the MRP problem, the following definitions apply:

S	$=$ Starting vertex $S \in V$,
G	$=$ Goal vertex $G \in V$,
v	$=$ current vertex $v \in V$,
$p(v)$	$=$ path from S to v,
v'	$=$ successor of v, $v \in V$, that is, v' adjacent to v,
OPEN	$=$ list of successors v' with paths $p(v')$
$c(v)$	$= g(S,v) + h(v,G)$, cost evaluation function
CLOSED	$=$ all v from OPEN that have been evaluated.
OPEN(v)	$= p(v)$, least cost path in OPEN

Using the definitions above, the A* algorithm from Section 3.6 can be extended to include the pseudocode for the MRP algorithm. The pseudocode for the enhanced A* algorithm for MRP becomes:

MRP Algorithm Pseudocode

```
1. put start $S$ on OPEN.
2. loop
   (a) remove from OPEN and place on CLOSED the least cost path $p(v) =$ OPEN($v$).
   (b) if $v = G$, $p(v) = p(G)$ is a solution, so exit with success.
   (c) determine all successors $v'$ of $v$.
   (d) if $v'$ not on CLOSED, put $v'$
   (e) else for each successor $v'$:
       i. Calculate actual cost $g(S, v')$.
       ii. Estimate cost $h(v', G)$ to goal.
       iii. If total cost $c(v')$ is less than CLOSED($v'$), then put on OPEN.
end MRP.    □
```

The algorithm successively increments the path length by one vertex starting with S and evaluating each successor v' until the goal G is reached. Each vertex removed from the OPEN list represents the least cost path from S to v. Therefore, when $v = G$, a least cost path from S to G is obtained. This assumes that there is at least one path from S to G, and that $c(v) > 0$, $\forall v \in V$. The purpose of the heuristic function $h(v', G)$ is to guide the search algorithm towards those vertices which are most likely to be on the optimal path. Intuitively, a good heuristic will reduce the number of vertices that must be evaluated before the goal is reached. Discussion of the heuristic is provided in Section 3.6.3.1, and details of the heuristic function used for this research are described in Section 4.5. The next step in the high-level design is to describe the MRP algorithm using a formal notation called UNITY in order to prove correctness of the algorithm and to identify the parallel aspects of the solution (high-level design).

4.3.2 Formal High-Level Design. A formal method called UNITY is used to provide a concise notation for specifying problems, proving correctness of design, and identifying concurrent aspects of the design. UNITY is a "theory — a computational model and a proof system ..." (20:8). It provides a formal means of program development and design for a variety of distributed and concurrent algorithms. Although a UNITY program is not executable, the formal logic in UNITY allows step-wise refinement to the lowest level of implementation needed to map to the target parallel architecture while verifying and maintaining program correctness along the way. A UNITY program does not terminate in the normal sense of the word, but achieves a state called "fixed point" (FP). A program reaches fixed point when execution of any statement leaves the current state unchanged (20:53). The following UNITY program for the MRP algorithm, and corresponding correctness proof, assumes the reader has an understanding of UNITY notation and formal logic. For more information on UNITY, the reader is referred to the text by Chandy and Misra (20), and the AFIT Compendium (30).

As stated earlier, the A* algorithm with heuristic function $h = 0$ is equivalent to a shortest path algorithm. Therefore, the analysis of shortest path algorithms in (20) can be augmented with a heuristic function to represent the MRP algorithm in UNITY, as shown in Figure 4.1.

```
Program MRP Search

declare
    Vertice: Integer                      {Point in the search graph.}
    $V$: set of Vertice                   {All points in the search graph.}
    PATH: sequence of Vertice
    OPEN, CLOSED : set of PATH
    min(OPEN), $P$ : PATH
    $S$, $G$, $v$: Vertice                {$S$ = Starting point; $G$ = Goal.}
    end($P$): Vertice $= v$               {Last Vertice in the PATH.}
    next($P$): Vertice $= v'$             {The next Vertice adjacent to end($P$).}
    $g(S, v)$ : real                      {Actual cost of travel from $S$ to $v$. }
    $h(v, G)$ : real                      {Estimated cost of travel from $v$ to $G$. }
    $C(v)$ : real                         {Total estimated cost $= g(S,v) + h(v,G)$. }
    cost($P$) : real                      {Path cost = $C$( end($P$))}

always
    {Increment path $P$ by 1, by concatenating $v'$ to $P$.}
    $\langle \| v' : v' \in V :: P_j = [P_i; v']$   if $((v' = \text{next}(P_i)) \wedge (\text{end}(P_i) \neq G) \wedge (i \neq j))\rangle$
 [] $\langle \text{cost}(P) = C(S, \text{end}(P))\rangle$
 [] $\langle \| P : P_i, P_j \in \text{OPEN} :: (P_i = \text{min(OPEN)}) \wedge (\text{cost}(P_i) \leq \text{cost}(P_j))\rangle$
    { Above states that min(OPEN) is always the least cost path.}

initially
    $P$ = [Start]
    OPEN = $\{P\}$
    CLOSED = $\emptyset$

assign
    $\langle P :=$ min(OPEN)                 {Get least cost path in OPEN.}
    {Put $P$ on CLOSED }
    [] $\langle$CLOSED := CLOSED $\cup P \;\|$
       {Expand $P$ with all adjacent points $v'$. }
      $\langle \| v' : v' \in V ::$ OPEN := OPEN $\cup [P; v']$ if $([P; v'] \notin$ CLOSED$)\rangle\rangle$   if (end($P$) $\neq G$)$\rangle\rangle$
       {Stop when Goal is reached, i.e. Fixed Point. }

end {MRP Search}
```

Figure 4.1 High-Level UNITY Design for the MRP Program.

4.3.2.1 Informal Description of Design. There is a finite number of vertices in the search graph such that $N = |V|$. Assume all the vertices are numbered $1..N$. Starting with S, adjacent vertices are evaluated one at a time and concatenated to the path P until the goal G is reached. The cost of the path is determined by the cost function $C(v)$. The new paths formed by concatenating vertices to P are put on the OPEN list. Least cost paths are taken off the OPEN list first. As the paths are removed from OPEN and evaluated, they are put on the CLOSED list. The heuristic function $h(v, G)$ determines the order that the paths are removed from OPEN. If $h(v, G) = 0\ \forall v \in V$, the problem becomes a single-source shortest path problem as described in (20:450-453) with complexity $O(v^2)$. However, if $h(v, G)$ is a good estimator of the actual total cost, the number of vertices to be evaluated may be reduced, thereby reaching fixed point much sooner and improving the overall efficiency of the algorithm. The optimal solution is obtained when the goal is reached. Also, the fixed point occurs when the goal is reached because all other paths on OPEN would have greater costs.

An examination of the UNITY description reveals the inherent parallelism of the problem. For each path P removed from the OPEN list, all new paths formed by adjacent points v' can be expanded concurrently. Also, the removal of P from OPEN, and the expansion of P can be performed concurrently if different (distributed) OPEN lists are used.

4.3.2.2 Formal Description of Design. In this section, the design is described using formal proof logic from UNITY. The invariant and fixed point are specified so that a proof of correctness of the design can be performed. The max size of CLOSED is equal to $N = |V|$ because CLOSED stores the least cost path from S to $v \in V$. Likewise, the max size of OPEN is limited to N because the CLOSED list is checked before new paths are put on OPEN. Initially, the path P from S contains only S, and the cost $c(S) = h(S, G)$ is completely estimated by the heuristic. As points are concatenated to the path P, the estimated cost is augmented by the actual cost

$g(S, v')$ from S to v'. Since paths are removed from OPEN in least cost order, $P = \min(\text{OPEN})$ is comprised of least cost sub-paths. Therefore, this design has the following invariant:

invariant

$$\text{cost}(P) = C(v) :: (v = \text{end}(P)) \wedge (P = \min(\text{OPEN})) \tag{4.10}$$

The fixed point holds when the goal is reached, because all paths left on OPEN have greater costs. Expressed in formal UNITY notation gives the following:

$$FP \equiv \langle (P = \min(\text{OPEN})) \wedge (\text{end}(P) = G) \rangle \tag{4.11}$$

A fixed point is reached if there exists at least on path from S to G, or when OPEN $= \emptyset$. The progress condition is that the path lengths are increased by successors (adjacent vertices) and the new paths added to OPEN or CLOSED if the state is not at fixed point (*i.e.* the goal not reached):

$$\neg FP \wedge (\text{OPEN} \neq \emptyset) \;\mapsto\; ([P; v'] \cup \text{OPEN}) \vee ([P; v'] \cup \text{CLOSED}) \tag{4.12}$$

Equations 4.10, 4.11, and 4.12 define the constraints for the strategy of the MRP UNITY design.

4.3.2.3 Proof of Correctness of Design. In this section, the constructs of UNITY are used to describe the proof that the design satisfies the requirements given in Equation 4.1.

Proof: Since the number of vertices not yet evaluated is always decreasing, the fixed point is eventually reached. At any fixed point reached in program execution, both the invariant (Equation 4.10) and FP (Equation 4.11) hold. Therefore, it is sufficient to show that when the goal is reached, the path is the optimal path.

Since all costs are ≥ 0, for every pair of vertices (S, v_i) there exists a least cost path from S to v_i if a path exists. The path P is a sequence of vertices $= [S, v_1, v_2, \ldots, v_{i-1}, v_i]$. The next path selected for evaluation is $P = \min(\text{OPEN})$. After evaluation, all new paths of length $|P| + 1$ have been added to the OPEN list. Since min(OPEN) is the least cost path in OPEN, the following holds $\forall P$:

$$C(v_{i-1}) \leq C(v_i), \quad v_{i-1}, v_i \in P \tag{4.13}$$

Consequently, all expanded paths $[P, v_i]$ put on OPEN have costs $\geq C(v_{i-1})$. Therefore, when end$(P) = G$, the costs of all paths remaining on OPEN are $\geq P$, and $P = Best$. $\square$

4.4 Object-Oriented Design

Object-oriented design (104) provides another means of specifying a problem in terms of object models. It provides a modular approach to encapsulating information about a data structure that is lacking in a functional design approach. The data objects are the Terrain, Vertices, Path, Radar, and Aircraft. The OPEN and CLOSED lists are data stores, or modelled as sets of Paths. The Paths in OPEN are ordered with respect to evaluation cost. The requirements specification (Section 4.1) gave a general description of the terrain object. The correctness of the design is not dependent upon the type of terrain object chosen, but further refinement of the design cannot be made without specifying the terrain object at this level. Based on the discussion in Section 2.3.1, the terrain model is a 3-D grid structure of Vertices. The Vertices are composed of X (latitude), Y (longitude), and Z (elevation) values. A Path consists of a sequence of Vertices, and has associated attributes of distance cost and radar cost. The object diagram for the Terrain, Path, and Vertice is shown in Figure 4.2. The two other data objects, Radar and Aircraft, are used in the calculations of radar cost, and have associated attributes as shown in Figures 4.3 and 4.4.

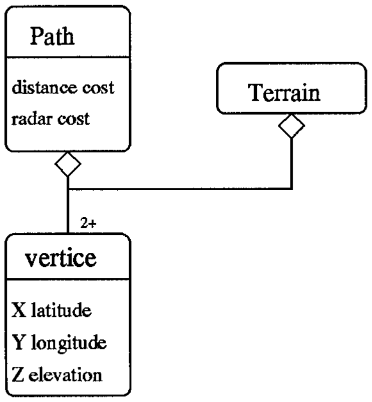

Figure 4.2 Object Model for a Mission Route

Radar
type: monostatic, bistatic
TxPower: Watts
TxGain: Integer (dB)
RxGain: Integer (dB)
Frequency: Hertz
MinAngle: Degrees
Location: Vertice

Figure 4.3 Object Model for Radar.

Aircraft
MaxRange: kilometers
MaxCeiling: meters
MinAltitude: meters
MaxTurn: degrees
RCS: meters^2

Figure 4.4 Object Model for Aircraft.

4.4.1 Dynamic Model. The dynamic model of the states described by the UNITY program is as shown in Figure 4.5. The dynamic model depicts the states and transitions of the program. The *searching* superstate (104:97) constitutes the main body of the MRP UNITY program. The *initial* state encapsulates the preparation, set-up, and initial assignments for the program to carry out the search process. The details of each state are described in the next chapter on low-level design and implementation. The *searching* superstate is composed of four states:

select min path the selection of the next path to expand.
expand path create new paths for each adjacent vertex to path and evaluate cost.
store on OPEN if new paths not on CLOSED, put on OPEN.
store on CLOSED put old path on CLOSED.

The transitions follow the pseudocode and UNITY program for the MRP search process: once the least cost path is *selected*, a path expansion is needed; once the path is *expanded*, the new paths must be stored; once the new paths are *stored*, another selection is required. If the endpoint to the selected path is equal to the goal vertex, a solution is reached and the event *solution found* transitions the program to a final state. Details describing each state are reserved for lower levels of abstraction provided in the next chapter.

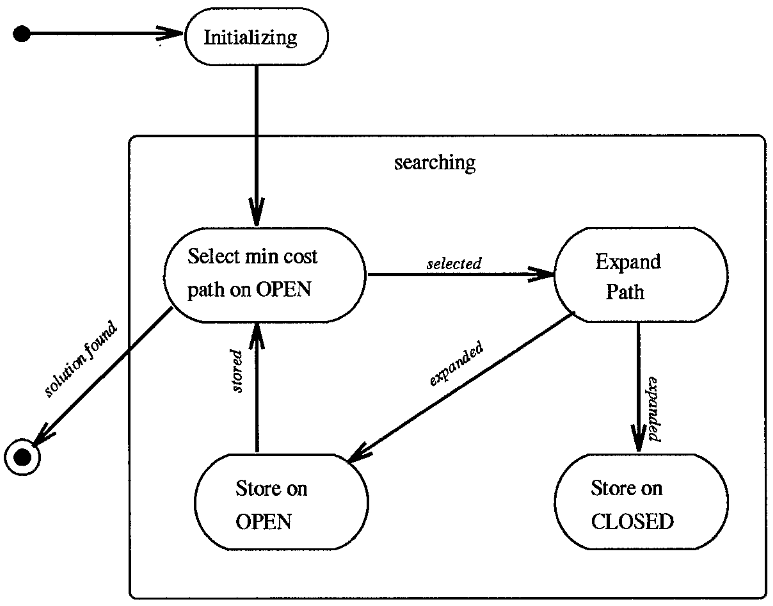

Figure 4.5 Dynamic Model for a MRP Algorithm

4.5 *Heuristic*

The heuristic must be monotonic for greatest efficiency of the A* algorithm (96:85). Two criteria, distance and radar exposure, are used in the evaluation of the actual "cost" of a selected path. However, as discussed in Section 3.6.3, only distance is used in the heuristic estimate to guarantee monotonicity. The distance heuristic is calculated using the Euclidean distance formula:

$$d = \sqrt{(X_P - X_T)^2 + (Y_P - Y_T)^2 + (Z_P - Z_T)^2} \tag{4.14}$$

The subscript P indicates the coordinates of the current point in the path P, and the subscript T indicates the coordinates of the target. With the aircraft turn model as described in Section 2.3.3, this heuristic provides an exact estimate of the actual distance of paths along an axis or diagonal. The greatest error in this estimate can be found by first determining the greatest error for two dimensions, and then extending it to three dimensions. For two dimensions, the greatest error lies somewhere between a diagonal and an axis. Using simple trigonometric relationships as shown in Figure 4.6, it can be shown that the greatest error in the heuristic estimate for two dimensions occurs when the straight-line distance between the Start and Target forms a 22.5° angle with an axis. In the figure, angle $E = 45°$, a is the heuristic estimate of the straight-line distance between S

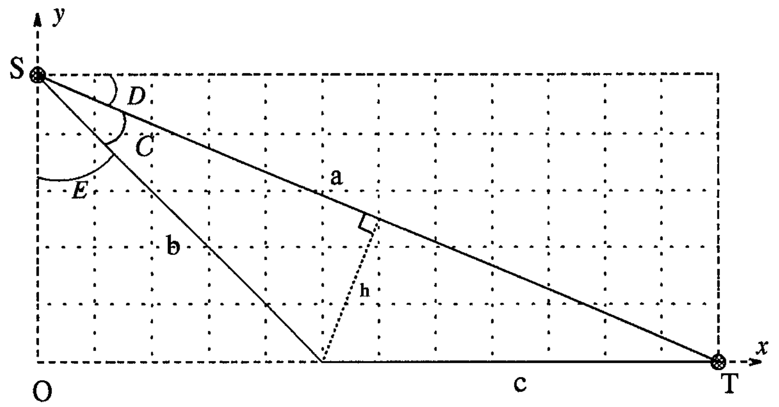

Figure 4.6 Distance Heuristic in Two Dimensions.

and T, and $b+c$ = the actual distance that can be plotted on a discrete grid structure. To determine the greatest error in the heuristic estimate, the maximum value for $b+c$ must be determined. From simple trig relationships, max$(b+c)$ occurs when $b = c$. The law of cosines is $c^2 = a^2+b^2-2ab\cos C$, which simplifies to $\cos C = \frac{a}{2b}$ when $b = c$. Let X be the distance OT, and let Y be the distance OS. Then the following relations apply:

$$a = \sqrt{X^2+Y^2} \tag{4.15}$$
$$b = Y\sqrt{2} \tag{4.16}$$
$$c = X - Y \tag{4.17}$$

Making the appropriate substitutions into the law of cosines yields the following results:

$$C = \cos^{-1}\sqrt{\frac{Y^2(1+\sqrt{2})^2+Y^2}{8Y^2}} \tag{4.18}$$
$$= \cos^{-1}\sqrt{\frac{2+\sqrt{2}}{2}} \tag{4.19}$$
$$= 22.5° \tag{4.20}$$

Therefore the max error for 2-dimensions is

$$\frac{2b}{a} = \frac{1}{\cos 22.5} = 1.0824 \tag{4.21}$$

This indicates that for two dimensions the heuristic estimate is within 8.24% of the actual shortest path on the discrete grid.

For three dimensions, the calculations are not as straight-forward, so the maximum error was determined empirically. The heuristic calculates 3-D distance according to Equation 4.14. For computing the shortest path error in 3-D, assume that $Z < Y < X$. The shortest distance on a discrete 3-D grid would be found by moving Z points along all axes, then $Y - Z$ points along the x and y axes, and the remainder $X - Y - Z$ along the x axis. Table 4.1 shows the ratios for different values of X, Y, and Z, and the max ratio for a $100 \times 100 \times 25$ grid is 1.1281. An examination of the data in Table 4.1 reveals that this 3-D distance heuristic never underestimates the actual distance by more than 12.81%.

X	Y	Z	Max Ratio
2	1	1	1.11535507
3	1	1	1.12525565
5	2	2	1.12525565
6	2	2	1.12525565
6	3	2	1.12547359
7	3	2	1.12754715
9	4	3	1.12769843
10	4	3	1.12790542
12	5	4	1.12799691
19	8	6	1.12807195
22	9	7	1.12808250
41	17	13	1.12809247

Table 4.1 Max Ratios of the Actual Distance of the Heuristic Estimate for Different Values of X, Y, Z.

4.6 Summary

In this chapter, the requirements specification and high-level design are developed for the MRP problem. The requirements specification captures the objectives of the problem through problem analysis. The primary objective is to find the least cost flight path for a single aircraft on a mission from a starting point to a target. A modified A* algorithm is selected as the heart of the search process because of its optimal performance with a monotonic heuristic. The UNITY language is used to provide a formal notation for describing the MRP algorithm, capturing the concurrent aspects, and proving correctness of the algorithm. Object-oriented techniques are included to further define the problem and capture the essential elements for the high-level design. The heuristic is shown to be a "good" approximation of the actual distance on a discrete 3-D grid, because it provided estimates within 12.81% of the actual distance along the discrete path. In the next chapter, refinements to the high-level design are described including details for lower levels of abstraction.

V. Low-Level Design and Implementation

5.1 Introduction

This chapter continues the refinement of the high-level design of the previous chapter. The low-level design and implementation associated with this research does not differ significantly from the previous research of Droddy (29). The "C" programming language was used in the previous research and was not changed for this research effort. The main difference from the previous research is in the low-level design and implementation of the A* algorithm. Droddy used a manager/worker strategy to control the search process, whereas this research found that elimination of the manager provided better performance. The changes made to the previous design account for approximately 13% of the entire code. Therefore, this chapter reports the low-level design changes made to update Droddy's approach to the new design (refer to Chapter 5 in (29) for details of Droddy's low-level design).

5.2 Low Level Design

The low-level design defines the structure of the code used to perform the mission route planning. The code had the following modular structure:

- *Host* program interface between user and the parallel processors.
- *Node* program which executes on the processors.
- Header files contain global constants and variables used by the *node* program.
- Input files for the terrain, radar, aircraft, and mission scenario.

Each of these are discussed in the following paragraphs:

Parallel Architectures. The iPSC/2, iPSC/860, and Paragon were the three parallel computers available for use in this research[1]. The iPSC/2 is a hypercube architecture based on i386DX (16 MHz) processors; the cube network has a message bandwidth of 2.8 MB/sec. The

[1]The iPSC/860 hypercube at Beaverton, Oregon, was used very early in the research, but a password error prevented continuation of the research on that platform.

iPSC/860 replaces the i386 processors of the iPSC/2 with faster i860XR (40MHz) processors, but uses the same communication network as the iPSC/2. The Paragon is the most current of the three Intel parallel systems available to AFIT researchers. It is a mesh architecture based on the i860XP (50 MHz) processor, and it has a reported message bandwidth of up to 175 MB/sec (62). The packet transfer time for the Paragon is so low — 40 nanoseconds per hop across mesh — that physical location of processors in the mesh becomes unimportant for parallel algorithm performance. For the two hypercube systems, however, the slower interprocessor communication makes processor location an important design consideration on those networks. The general architecture structure of each system is shown in Figure 5.1.

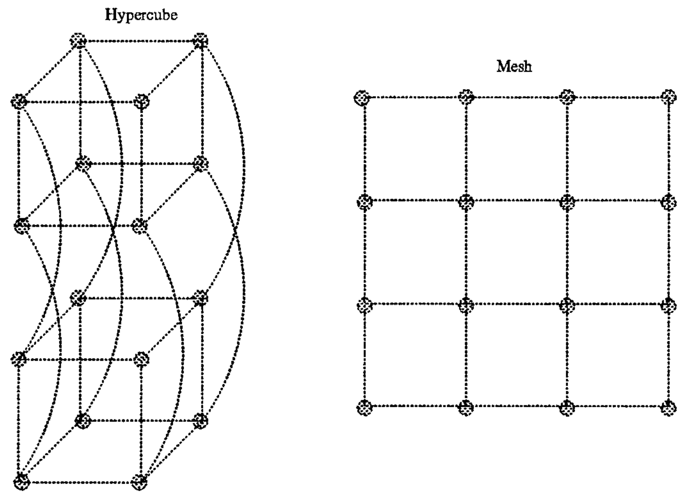

Figure 5.1 Comparison of the General Structure of a Hypercube vs a Mesh Parallel Architecture with 16 Nodes.

Host Program. Each system has specific commands to load programs onto the parallel processors (61, 63). Droddy developed a special interface called a *host* program that consolidated the load commands and performed I/O checking to load the A* search program onto the iPSC/2 and iPSC/860. Slight modifications were made to the *host* to be compatible with the Paragon. These changes pertained mainly to syntax differences between the two architectures. The task structure of each version remained the same. The tasks comprising *host* are:

1. read the names of the input files
2. check that input files exist
3. get the appropriate number of parallel nodes
4. load programs on the nodes
5. wait for the nodes to finish
6. release the parallel nodes

These tasks are part of the initialization of the program and are sequential in nature. This initialization takes a constant amount of time because the number of input files remains constant for all test cases.

Node Program. This program is the same for the Paragon mesh, and the iPSC/2 and iPSC/860 hypercubes. The *host* provides the interface that allows the same program to be used on either the hypercube or the mesh. For the *node* program, the main difference between the final design of this investigation and that of Droddy (29) is in the implementation of the A* algorithm (see Section 3.6). Droddy used a centralized OPEN list controlled by a manager, and local OPEN lists at each worker processor. A CLOSED list was not implemented. However, this research found that distributed local OPEN lists and a global CLOSED list performs significantly better and eliminates the need for a manager processor. As part of the initialization of the *node* program, one worker processor generates unique paths and distributes them to the other workers. After that point, the tasks performed by each worker are identical. The high-level design from the previous chapter (Section 4.3) is refined for the *node* program to add specific detail for the A* implementation of MRP. The control flow between the *host* and *node* programs is shown in Figure 5.2. The tasks for the *node* program are decomposed as follows:

1. Initialization
   (a) Receive the input data/parameter files from the *host* program
   (b) Read the contents of the files into internal data structures
   (c) If I am worker 0, generate and send initial paths to other workers
   else receive initial paths from worker 0.
   (d) Calculate an initial cost bound for the best path P_{best} to the target.

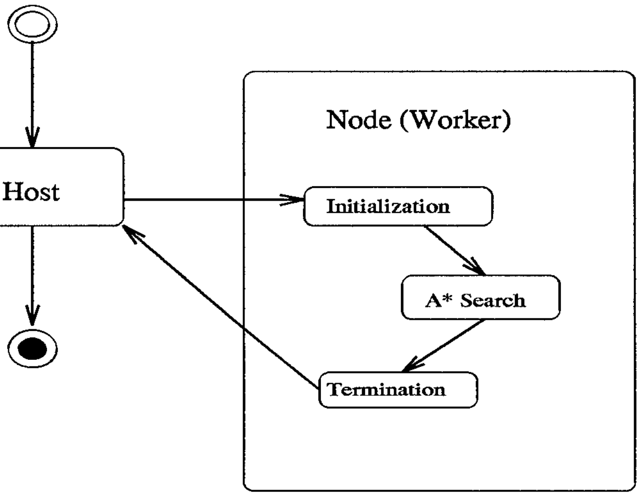

Figure 5.2 The Control Flow Between the *host* and *node* Programs.

2. Perform A* search.

- Begin Loop.
  (a) Remove path P_i from OPEN.
    - If P_i already on CLOSED, discard and remove another path from OPEN.

  (b) Put on CLOSED and broadcast to other workers.
  (c) Generate new paths P_j from P_i and put on OPEN.
  For each P_j do:
    i. calculate actual distance cost to the current point.
    ii. calculate actual radar cost to the current point.
    iii. estimate remaining distance cost to the target.

  (d) If target reached, update P_{best} and broadcast this to other workers.
  (e) If received P_{best} info from other worker, update P_{best}.
  (f) If my OPEN list empty, request work from other workers.
  (g) Receive CLOSED path information from other workers and update my CLOSED list.
  (h) If work request received then
    - if my OPEN list not empty, send paths to requestor;
    - else if my OPEN is empty and I am not the requestor, pass on the work request to next worker in the ring.
    - else if I am requestor and my OPEN is empty, broadcast terminate message and exit loop;

  This last step will occur when the target has been reached, and all workers have emptied their OPEN lists.
- End Loop

Different functions and procedures are implemented for each task. The function hierarchy used in the *node* program is shown in Figures 5.3 and 5.4. The functions are described in the following paragraphs:

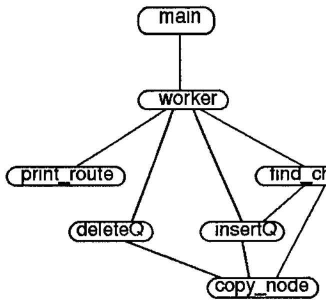

Figure 5.3 Software Function Hierarchy (Part 1)

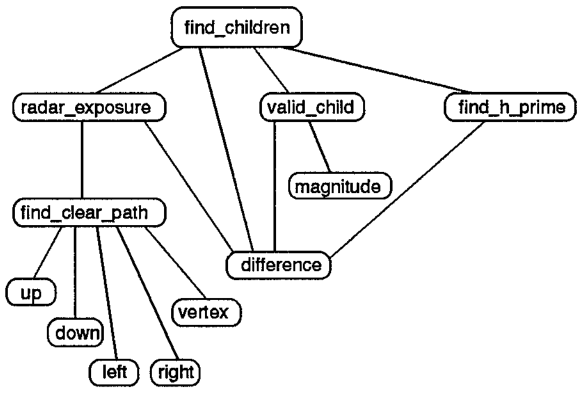

Figure 5.4 Software Function Hierarchy (Part 2)

InsertQ and DeleteQ. These functions, in conjunction with the variable *Qlength*, provide a linear priority queue abstract data type (ADT) (118:173) which stores the generated paths. *InsertQ* begins at the front of the queue and proceeds through until it finds a path of equal or greater cost, then inserts the new path into the appropriate position. The *insertQ* function is an order $O(k)$ operation for k paths on the OPEN list because it must potentially make k comparisons to determine the cost priority of the new path that is being inserted. The maximum size of the queue is fixed, so $k \leq$maxQsize. *DeleteQ* removes the least-cost path from the front of the queue. It is an order $O(1)$ operation because it always removes the front element of the priority queue.

Both *insertQ* and *deleteQ* functions update the variable *Qlength* to keep track of the total number of paths on the OPEN list.

Copy_node. This utility function *copy_node* is used to copy path information for manipulation by the *insertQ* function. Each "node" represents a path implemented as an array of 200 gridpoints (X,Y,Z coordinates). The *copy_node* function copies only the nonempty points in the path and does not search through every array position. Therefore, this function takes $O(n)$ time where $n \leq 200$ is the number of gridpoints in the path.

Print_route. This is an I/O function that prints the path points, costs associated with each point, and total distance and radar costs for the optimal path selected.

Find_children. This function is the "heart" of the worker program. It generates all potential children for a given path and inserts them on the OPEN list. A child of a path is that path formed by concatenating an adjacent point to the last point of the current path. This function invokes several other functions to determine the children of the current path.

Valid_child. This function is called by *find_children.* When *find_children* generates a potential child path, *valid_child* determines the validity of the potential child by the following set of rules:

1. it is within the X, Y limits of the terrain representation
2. it is within the altitude or Z limits established by the terrain (lower altitude limit) and the representation (upper limit)
3. it is within altitude limits set for the mission (in the ATO)
4. it is within the aircraft combat radius
5. it is within the aircraft turning capability

The rules are checked in the sequence shown and are applied using the models described in Section 2.3 and in Section 4.4. If the child path being evaluated meets all the rules, the child is considered a valid child. If any rule is not met, the child path is not valid, and is discarded.

Find_h_prime. The function *find_h_prime* implements the heuristic function which estimates the remaining cost of reaching the goal from the current path endpoint. The h' is calculated to be the simple straight-line distance to the goal; radar is not factored into the estimate. This heuristic is monotonic. If radar costs were included in the estimate, the heuristic would not be monotonic (see Section 3.6.3 for more discussion).

Radar_exposure. This function calculates the radar exposure at a given point. The radar exposure is determined by the aircraft RCS, the radar location and characteristics, and the distance from the given point to the radar. In the previous research (29), the multiple radars were treated as both monostatic radar transceivers and bistatic transmitters and receivers. However, since the bistatic RCS is different from the monostatic RCS, this function was modified so that it only calculates monostatic radar exposure. This design was improved by incorporating a minimum detectable angle for the radar (106). This minimum angle is common to all radar systems and is determined by the surrounding terrain and physical construction of the radar. Knowing the minimum angle of the radar allows the MRP algorithm to plan routes that fly "under" radar to avoid exposure to radar. The exposure calculation is cumulative with respect to the number of radars within line-of-sight of the given point. This results in an $O(r)$ complexity for r radars. To determine if the aircraft is visible (*i.e.* within line-of-sight) to each radar site, the function *find_clear_path* is used. This is done before the radar exposure calculation so that if the plane is not visible from t radars, the radar exposure calculation is reduced to $O(r - t)$ complexity.

Find_clear_path, Vertex, Up, Down, Left, Right. The function *find_clear_path* is implemented using a ray-tracing algorithm from Glassner (43:29). The other five functions are used to average altitude when traveling from one vertex to another; that ability is not implemented in the current version of this program. However, the functions are retained to permit their inclusion in future development.

Difference. This function calculates the straight-line distance from one point to another. It is another utility function employed by a variety of functions in the program.

Magnitude. This function is slightly different from the previous in that it is given the X, Y, Z components of a vector and calculates the magnitude of the vector. It could be simulated by providing the vector as one point and $0, 0, 0$ as the other when invoking *difference*, but that would reduce the clarity of the program.

5.3 Design Issues

Some important issues remain in determining the detailed design and implementation of the MRP program. They are discussed in the following sections:

5.3.1 Mapping the Software Architecture to a Hardware Platform. The old design was mapped to a hypercube structure for the iPSC/2 and iPSC/860 parallel computers. A discussion of this mapping is given by Droddy in (29). The main design goal for mapping to a given parallel architecture is to minimize the communication delays for message passing between processors. The hypercube systems used in the previous research allowed some control over the physical configuration of the processors in the hypercube. In contrast, the Paragon system used in the current research did not allow control of the configuration of the mesh. The shape of the mesh for a given number of processors could vary from run to run depending on the configuration of the other users of the system. The user did not have control over this configuration. However, communications between the Paragon processors was greatly improved over the older hypercube systems. Each processor "node" on the Paragon is actually two i860 RISC[2] processors with one dedicated to communication under normal operation (62). This researcher did not investigate and quantify the effect of the Paragon message-passing protocols on the performance of this code design. That requires de-

[2]Reduced Instruction Set Computer.

veloping specific experiments to characterize the communication performance under varying loads and scenarios. That was left as a topic for future research.

5.3.2 Data Structures. When defining data structures, memory constraints for the target system must be considered. For example, the AFIT iPSC/2 (only eight nodes) provided 12 MB/node of available memory compared with only 8 MB/node on the Beaverton iPSC/860 (64 node). The Paragon used in this research provided the greatest amount of available memory with 24 MB/node. The executable code for the current design requires only 0.5 MB of memory. Therefore, most of the available memory is used to store the data structures. The header file compiled with the *node* and *host* programs contains the constants which specify the maximum size of the array data structures. To make maximum use of memory on a given system, the constants specified in the header file are tailored so that the maximum size of the data structures fit into the available memory.

Terrain Representation. Terrain is represented by a three-dimensional grid[3] of XYZ coordinates corresponding to latitude (X), longitude (Y), and elevation (Z). The scale factor along the X axis is the same as along the Y axis, and is twice that of the Z axis: Xscale = Yscale = 2× Zscale. This was done to provide greater resolution for elevation than for the XY plane. The terrain data structure was chosen to be an array so that point values could be accessed in $O(1)$ time. For this research, a 100×100 array of elevation values was selected for the terrain model. The elevation values are stored as 4-byte integers, so the memory required for the terrain is $100 \times 100 \times 4 = 40000$ bytes = 39 KB[4]. This terrain data is read from a file during program initialization and is stored locally by each processor. The elevation data affects the minimum altitude of path points selected by the program, and is also used in determining radar exposure.

[3]Alternatives to this representation are discussed in Section 2.3.1
[4]1 KB = 1024 bytes.

Path Record. The goal of the MRP program is to find the "optimal path" to the target. In the process, many paths are generated and stored on the OPEN list. The memory required for the path determines the maximum size of the OPEN list. Therefore, the path data structure should be designed for memory efficiency. The following is the C-language definition used in the MRP program header file:

```
typedef struct {
   int     number;                  /* Number of entries in the route  */
   US      x [MAX_PATH_LENGTH+1]; /* Vector of x locations           */
   US      y [MAX_PATH_LENGTH+1]; /* Vector of y locations           */
   US      z [MAX_PATH_LENGTH+1]; /* Vector of z locations           */
   int     VectorX;                 /* Direction vector in x direction */
   int     VectorY;                 /* Direction vector in y direction */
   int     VectorZ;                 /* Direction vector in z direction */
   float   distance;                /* Cumulative distance of the route */
   float   radar;                   /* Cumulative radar detection cost  */
   float   g;                       /* Cost of the given route         */
   float   cost;                    /* Calculated cost (f') of route   */
   int     link;                    /* Forward Links for the OPEN list */
} PATH;
```

The x, y, and z values form an array of vertices, stored sequentially. The MAX_PATH_LENGTH for this research was set to 200. This would allow the possibility of long circular paths around radars and through the "back-door" to the target[5]. The *link* value is a pointer used to order the paths stored on the OPEN list. The cost includes the heuristic estimate h' that is calculated but not stored. The x, y, and z arrays contain the XYZ point coordinates represented by unsigned short (US) integers which require two bytes. Integers and float values require 4 bytes of storage. Total memory required per path is therefore $(3 \times 201 \times 2) + (9 \times 4) = 1242$ bytes.

OPEN List. This is the most important data structure because all generated paths must be stored and evaluated to guarantee optimality of the final solution. The OPEN list is implemented as an array of Path records. The array size can be adjusted up or down to optimize the fit for a particular hardware platform. The maximum size is constrained by the bytes per Path

[5]The maximum path length along a straight diagonal from (0,0,0) to (100,100,25) for the current implementation is 144 points.

and the memory available per processor. The old design used a static array to store the paths in a prioritized sequence, and set the max size of the OPEN list to 4000 paths. Since each path required 1242 bytes, the old design needed 4.74 MB[6] of memory. The Paragon used in this research provided enough memory per processor so that the max size of OPEN was increased to 12000 paths = 14.2 MB of memory.

The abstract data type for the OPEN list was a priority queue (26:149-151). Paths are inserted and prioritized on the queue based on the cost (f') of the paths, with the least cost paths to the front of the queue. The *insertQ* operation uses a simple sequential search to determine the appropiate place for a new path. The *link* attribute of the path is used to point to the next path in the queue. Once the placement of the new path has been determined, the link attributes of the preceding path and the current path are updated to reflect the new ordering of the paths. Therefore, the *insertQ* operation requires $O(k)$ time where $k \leq$Qmax is the number of paths in the queue. The *deleteQ* operation removes the front of the queue in $O(1)$ time. A heap structure could be used to replace the simple sequential design, but this would change the times for the insert and delete operations. They would both become $O(\log k)$ time. It is not evident whether this would improve the execution time of the program, since the ratio of insert operations to delete operations is problem dependent.

CLOSED List. An explicit CLOSED list was implemented in this research. The data structure chosen for the CLOSED list was a 3-D array of path cost values. The array indices corresponded to the X, Y, and Z coordinates of the gridpoints. The X, Y, and Z max values were 100, 100, and 25, respectively. The cost values were stored as 4-byte floats. The total memory required for this implementation is $100 \times 100 \times 25 \times 4 \approx 1$ Mbyte. Since an array was used for this implementation, updating the cost value for any gridpoint takes constant $O(1)$ time.

[6] 1 MB = 1,048,576 bytes.

5.3.3 Input Data Files. To maintain maximum flexibility and follow "good" software engineering principles, key data items are input from files, rather than being "hard coded" into the program. This data is:

- terrain data
- radar characteristics
- the mission Air Tasking Order
- aircraft characteristics
- algorithm parameter information

This modular structure was used in (29), and was also employed for this research. This information is described in high-level terms in Section 2.3 and Section 4.4. The low-level details are provided in the following paragraphs:

Terrain File. The terrain file contains three integers to specify the size of the search space for the problem in the X, Y, and Z dimensions. Next is an integer specifying the scale factor (in meters). The scale factor is for the Z axis. The scale for the X and Y axes is equal to $2\times$ Zscale. The rest of the file consists of integers representing the elevation (mean sea level) for each of the X, Y vertices in meters. The scale was 250 meters, with X and Y set to 100 points. Other scale factors could be used to give more or less path resolution (see discussion in Section 2.3.3). For the given resolution, paths are planned over a 2500 square kilometer area.

Radar File. This research effort assumed that all radars had the same characteristics. This simplified the type and quantity of radar information to be stored. Different radar sites could be specified by giving one value for transmit power, antenna gain, and wavelength, and separate values for coordinates and minimum detectable angle. This information would be separately specified for each radar in an operational MRP system. In addition, a separate boolean variable would be needed to flag bistatic radars versus monostatic radars. The radar file contains the following information:

- number of radar sites for this problem

- the transmitter power
- the transmitting antenna gain
- the receiving antenna gain
- the transmitted signal wavelength
- for each radar site:
  - the X coordinate
  - the Y coordinate
  - the Z coordinate
  - the minimum effective angle (azimuth) for the radar

This information is used by the function *radar_exposure* to calculate radar costs for each point using the equations described in Section 2.3.2.1.

ATO File. The ATO file specifies the following information (29:5-15):

- the mission designator
- the X, Y, Z coordinates of the base location
- the X, Y, Z coordinates of the target location
- minimum flight altitude
- maximum flight altitude
- altitude type (mean sea level or above ground level)

Plane File. The plane file specifies the characteristics of the aircraft. This data is not classified since no specific aircraft was modeled. In an operational system, the data in this file would be classified (for most military aircraft). The file contains (29:5-15):

- aircraft radar cross-section (RCS)
- minimum turn angle
- maximum altitude at which the aircraft can operate
- maximum combat radius of the aircraft

In a fully developed system, the RCS information would comprise more than a single value. As discussed in Section 2.3.2.1, the simplification being made for this research is to treat the aircraft as a simple sphere for RCS purposes.

Algorithm File. This file specifies the following algorithmic parameters:

- Multiplication factor for radar cost.
- Weighting of radar detection

The criteria weight for distance traveled is one minus the weight assigned to radar. If a non-additive multicriteria function were used, or if three or more criteria were considered, additional values could readily be added.

Use of a file to specify these parameters permits easy experimentation, since values can be changed without recompiling the program. In a real-time system, default values could be hard-coded. The parameter values could then be varied as needed by the program.

5.4 Parallel Performance Issues

In addition to detailed design decisions for implementing a sequential algorithm, there are certain performance issues that are unique to parallel algorithm design. They are scalability, communication overhead, and load balancing. These are discussed in the following sections:

5.4.1 Scalability. The scalability of a parallel algorithm is a measure of its ability to achieve performance proportional to the number of processors (75). Two measures of scalability described in (75) are speedup, and efficiency. Speedup S is defined as the ratio of the serial run time for the best sequential algorithm compared to the execution time of the parallel algorithm applied to the same problem. $S = T_s/T_p$. Efficiency is a measure of the fraction of time for which a processor is fully utilized, and is defined as the speedup divided by the number of processors: $E = S/p$. The general speedup equation for the previous manager/worker design can be derived as follows:

$$\begin{aligned} S &= \frac{T_s}{\left(\frac{T_s}{p-1}\right) + T_m + T_i} \\ &= \frac{p-1}{1 + \left(\frac{(p-1)T_m}{T_s}\right) + \left(\frac{(p-1)T_i}{T_s}\right)} \end{aligned} \tag{5.1}$$

where

p	$=$	number of processors,
T_s	$=$	execution time for the sequential algorithm,
T_m	$=$	time for message passing of the parallel algorithm,
T_i	$=$	idle time caused by synchronization delays between the manager and workers.

The ratio $\frac{T_s}{p-1}$ is the serial time distributed over $p-1$ workers and one manager[7]. The manager is dedicated to managing the OPEN list and does not contribute directly to the search performed by the $p-1$ workers. The sequential time, $T_s = O(n^2)$, corresponds to the complexity of the problem as discussed in Section 2.4. The message passing time per processor T_m is determined by the number of messages and the transfer time per message (75:66). The number of messages is $O(\frac{n}{p-1})$ corresponding to the number of points to be searched per worker processor. The longest message transfer time is for the CLOSED list because it requires a broadcast to all processors. For a mesh with cut-through routing, the one-to-all broadcast takes $O(\sqrt{p})$ time (75:77). Therefore, the message passing time is

$$T_m = O\left(\frac{n}{p}\right) \times O(\sqrt{p}) = O\left(\frac{n}{\sqrt{p}}\right). \tag{5.2}$$

The idle time T_i is determined by the number of messages for the OPEN list that are sent between the worker and manager. In the manager/worker design of the old code, the manager receives updates to the OPEN list from all the workers. The time to send one message from a worker to the manager is $O(1)$ on a mesh with cut-through routing (75:66). The number of messages corresponds to the number of points to be searched, which is $O(\frac{n}{p-1})$ per worker processor. The manager collects an update to the OPEN list from each of the $(p-1)$ workers, and then sends a new search point from the OPEN list to each worker. The worker must wait from the time it sends an update message to the manager, until it receives a reply from the manager. This takes $O(p-1) + O(p-1) = O(p)$ since the $O()$ notation ignores the constants. Multiply this by the

[7]This assumes that the parallel algorithm performs at least the same amount of work as the sequential algorithm.

number of points to be searched to get the total idle time:

$$T_i = O\left(\frac{n}{p}\right) \times O(p) = O(n). \tag{5.3}$$

Substitute Equations 5.2 and 5.3 into Equation 5.1 and rearrange terms to get the following speedup equation for the manager/worker design:

$$S = \frac{p-1}{1 + O(\frac{\sqrt{p}}{n}) + O(\frac{p}{n})} \tag{5.4}$$

This reveals how the ratio p/n affects the speedup for the old manager/worker design. The number of points n is determined by the distance L between the starting point and the target for an MRP scenario, such that $L^3 < n$. For small L, as p increases, the denominator becomes larger which reduces the amount of speedup. Equation 5.4 shows that the manager/worker design may perform faster with larger problems than for smaller problems. However, actual values for speedup are dependent upon the terms hidden in the $O()$ notation. For example, $1000(p + \sqrt{p} + \log p) = O(p)$, so the hidden terms may have a significant affect on actual performance.

The factor that has the greatest affect in the denominator is the message transfer time which is determined by the architecture of the parallel computer. For the manager/worker design, the message transfer time dominates the speedup equation for large p. If it grows large enough, the speedup may become less than one, which means that the parallel version can perform worse then the sequential version. This is more easily seen if Equation 5.4 is rewritten with n as a constant:

$$S = O\left(\frac{1}{\frac{\sqrt{p}}{p} + 1}\right) \tag{5.5}$$

The new design eliminates the manager, and also eliminates the idle time caused by synchronization. Thus, the $p - 1$ term in Equation 5.1 becomes p, and the idle time T_i is eliminated. The resulting speedup equation for the new design is:

$$S = \frac{T_s}{\frac{T_s}{p} + T_m} \tag{5.6}$$

Rearranging terms, produces the following relation:

$$S = \frac{p}{1 + \frac{pT_m}{T_s}} \tag{5.7}$$

For the new design, messages are broadcast one-to-all, which has time complexity of $O(\sqrt{p})$ for each broadcast on a mesh (75:77). There is one broadcast for each CLOSED operation, and there are $O(\frac{n}{p})$ number of CLOSED operations (potentially one operation per point in the search area) per processor p. Combining terms yields: $T_m = O(\frac{n}{\sqrt{p}})$. The time complexity of the problem is $O(n^2)$ as shown in Section 2.4. Substituting all these relations into Equation 5.7 yields the following:

$$S = \frac{p}{1 + \frac{O(n/\sqrt{p})p}{O(n^2)}} = \frac{p}{1 + O\left(\frac{\sqrt{p}}{n}\right)} \tag{5.8}$$

This equation shows a significant improvement over the manager/worker design. The new design is less affected by message transfer times, so it reflects better speedup as the number of processors is increased. If Equation 5.8 is rewritten with n constant, the speedup for the new design is

$$S = O(\sqrt{p}) \tag{5.9}$$

This can be interpreted as a possible lower bound for the speedup of the new design. However, actual speedup values are dependent upon the parallel computer architecture, and the problem instance.

Communications Concerns. Messages between processors should be kept relatively small to reduce communication delays. Results of message-passing experiments (30) show that when message length exceeds 1024 bytes, the message is divided into multiple packets. This increases the communication overhead. In the current design, the message that contains the information on the CLOSED paths is the largest message. The entire path information is sent in the message. This could exceed 1 KB. The message size could be reduced by sending only the endpoint of the path, and the cost information. This would reduce the message size to less than 100 bytes. This was not implemented, so it is a topic of future research.

Load Balancing. In the previous design, each processor was terminated one by one when it finished its searching. The old design did not have an efficient load balancing strategy for termination of the algorithm. In the new design, the work is "evenly distributed" among the processors so that there is no idle time before the target is reached. Once the target is reached, dynamic load balancing occurs when a worker empties its OPEN list. Work requests are passed around the ring of processors, and those workers that have paths remaining on their OPEN lists will send some of their paths to the requestors. This allows each processor to work independently of the other. Synchronization among the processors is performed only when a worker receives its own work request, indicating that all the OPEN lists are empty, and the algorithm may be terminated.

5.4.2 Choice of Heuristic. A heuristic embodies problem-specific information to speed the search (96). If the heuristic is monotonic, the amount of searching is reduced (96:85). Grimm's heuristic was not monotonic and was time consuming to calculate (50). Run-time performance was poor as a result. Droddy developed a monotonic heuristic based on distance alone (29). Radar was not included in the heuristic because adding it would not guarantee monotonicity (see discussion in Section 3.6.3. In addition, a heuristic using only distance calculations could be made simply and quickly in constant $O(1)$ time.

The ideal heuristic would be one that could be "calculated quickly" that would provide an estimate of remaining costs that was very close to the actual costs. Using only one criteria (distance) for the heuristic guarantees monotonicity and is simple to calculate, but it is a gross underestimator of the actual cost when radar is encountered in the path. The new design therefore relies on the distributed OPEN lists and global CLOSED list to provide the search "pruning" in compensation for the underestimation given by the heuristic.

Initial Path Bound. An initial cost bound is placed on all paths inserted on the OPEN list. The bound is determined by calculating the minimum distance to the target, and multiplying this by the radar cost factor. This method is easy to implement, can be done in $O(1)$ time, and places an upper limit on the cost of the optimal path selected.

Handling Queue Overflow. The OPEN is implemented as a priority Queue static array. As such, the maximum size of the array must not be exceeded. In the current design, when the queue is full, the last 50 paths in the queue are discarded. For small test scenarios, the queue does not fill up. However, for larger problems, the number of paths generated exceeds the capacity of the OPEN list. When this happens, paths are discarded from the OPEN list, and optimality may be lost. As it turns out, the load balancing and partitioning of the new design is such that the OPEN list never became full when 8 or more processors were used.

5.5 Implementation Details

The implementation details concerning the coding standards, file structures, and parameter values are essentially unchanged from previous research conducted by Droddy (29). The changes required to incorporate the new A* algorithm design amounted to less than 13% of the 1900 lines of code.

5.6 Summary

This chapter continues the development of the design begun in Chapter 4, and provides the low-level design and implementation. Key concerns encountered in these design phases are discussed, particularly the issues of minimizing time and space complexity. The low-level parallel design includes a *host* and *node* program. The *host* is the interface between the user and the parallel network, and provides portability of the *node* program between the hypercube (iPSC/2 and iPSC/860) and mesh (Paragon) parallel systems. The *node* performs the MRP search which incorporates the new parallel A* design. The OPEN and CLOSED lists are the most important data structures of the low-level design. The CLOSED list is implemented as a 3-D array for fast $O(1)$ access, but this requires 1 MB of processor memory. The OPEN list is also implemented as an array, but it requires 1.2KB per path, or 14.2 MB for 12000 paths. The memory per processor for the Paragon is large enough (32 MB) for this design, but for the hypercubes (iPSC/2: 12MB; iPSC/860: 16MB), the size of the OPEN list must be scaled back. The new design is also shown to have potentially linear speedup, which is a significant improvement over the previous manager/worker design. The next chapter describes the tests of this design and contains the results and analysis of the experiments conducted using the MRP program.

VI. Experiments, Redesign, and Results

6.1 Introduction

This chapter discusses the experiments conducted to determine if the two primary problem objectives have been met:

1. Effectiveness: Select an optimal route from base to target through defended 3-D terrain.
2. Efficiency: Find the optimal route in 45 minutes or less for very large input[1].

The second objective is the real-time constraint imposed on the problem. Since the problem has been simplified for this research, the goal for the new MRP algorithm design is a factor of 10 improvement, or 4.5 minutes, for the simplified MRP problem. Experiments must be conducted to assess the performance of the program design tested against the specification (problem requirements). Although examples do not prove correctness of the program, they can be used as benchmarks for performance and provide a measure for determining if requirements have been achieved. Initial experiments are conducted on the program developed by Droddy (29) to validate results reported in (29). The experiments uncovered problems with Droddy's design, so his design is revised. This revised design could not satisfy objective 2, so a redesign is performed that incorporates a new parallel A* implementation. The redesign shows promise of satisfying all requirements. The following sections give the details and results of the experiments conducted. First, the metrics used to quantify the results of the experiments are described. Next, the results of experiments are given in three sections: the original code, a modified (improved) version, and a redesign incorporating a new parallel A* algorithm.

6.2 Metrics

Two metrics are predominantly used in these experiments. They correspond to the two primary research objectives listed in the introduction (optimality of the selected route, and execution

[1]Reference Section 2.2.

time). The experiments are designed in order to "accurately" determine these metrics. Metric 2 is the easier of the two to measure. Optimality of the selected route is more difficult to validate because it required hand calculating the distance and radar costs along the path. This is the first problem discovered with the previous research. In (29), there was no independent method used to validate the optimality of the route selected by the program. Optimality was determined simply by repeatedly executing the code on a given scenario to determine if the same solution is obtained each time. This does not guarantee that the optimal solution was obtained. Since the complexity of the terrain representation directly affects the calculations of cost to validate optimality, the terrain models used in the previous research were examined. Figure 6.1 shows one of the two terrain models used (not drawn to scale). The terrain was artificially created. Although it looks regular, a closeup

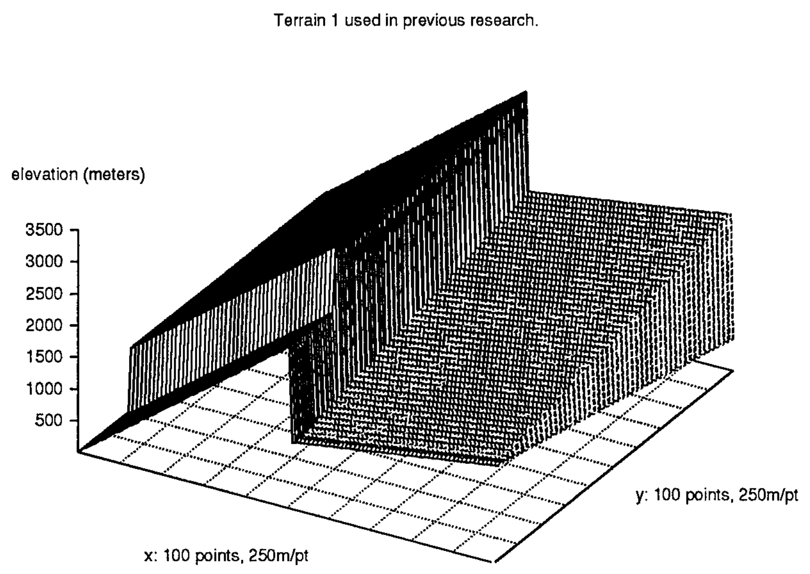

Figure 6.1 Artificial Terrain Model Used by Droddy (29).

shown in Figure 6.2 reveals unnatural ridges in the terrain. In addition, the data file for that terrain showed that the gap between the ridges was extremely narrow (only one discrete point[2] apart), and extremely deep (eight discrete points in depth). Terrain model #1 was unrealistic and overly

[2]The scale factor per point was 250 meters.

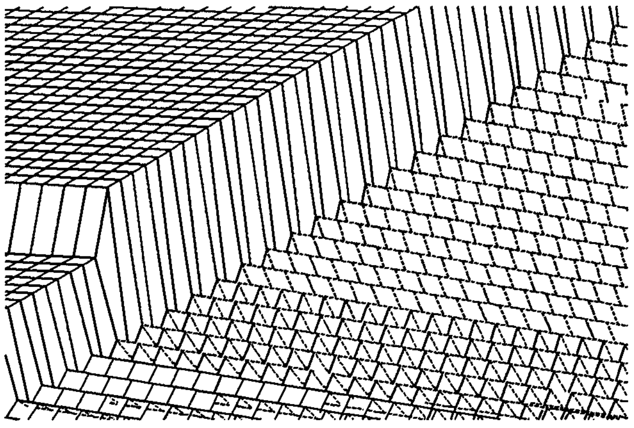

Figure 6.2 Expanded View of Terrain Showing Unrealistic Ridges.

complex, and therefore made it difficult to independently validate optimality of selected routes. The second of two models used was less suitable than the first. It is plain to see from Figure 6.3 that it would be extremely difficult to hand calculate route costs using this model. In addition, selecting the starting point, target, and radar positions required examining the data file to ensure that the placements were not "under" the terrain.

A new terrain model had to be developed that was uncomplicated so as to simplify validation of route optimality, yet suitable to provide realism for the route selections. But first, experiments were performed with the previous code to evaluate the execution time for various scenarios. This is discussed in the next section.

6.3 Tests Performed on Old Code

Droddy noted in Section 7.8.3 of (29) that his code did not always terminate, but he didn't explain why, or under what circumstances. Therefore, the first experiments were conducted to document the conditions that would cause the program to fail to terminate.

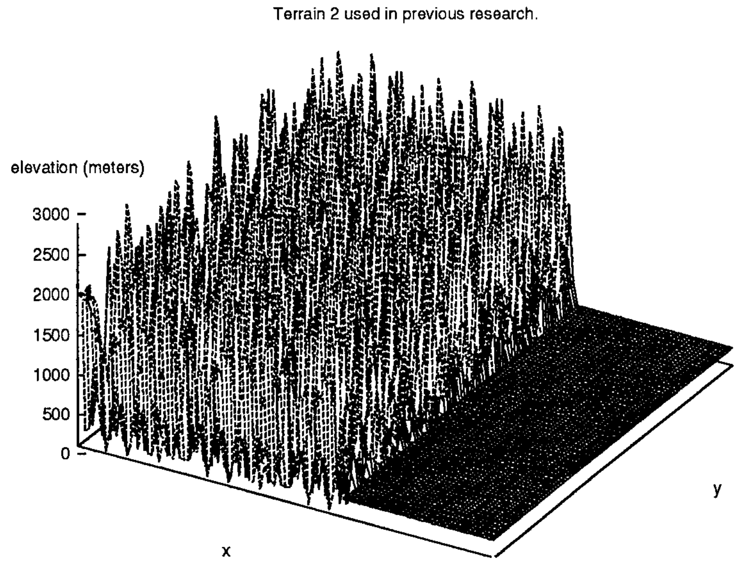

Figure 6.3 Second Terrain Model Used in Previous Research (29).

6.3.1 Experiment 1: Document Behavior of Old Code. This experiment is actually a group of experiments which repeat those performed by Droddy on the old code using the original data files. The tests are performed on the 8-node Intel iPSC/2 hypercube at AFIT. The mission scenarios varied the placement of the starting point, target, and up to 5 radars on both terrain 1 and 2.

Results: For approximately half of 50 combinations of scenarios, the code did not terminate. For some scenarios, the code ran for 28 hours without successful termination. Of the 25 that completed, execution times were three minutes or less. The input files were analyzed to determine if there was a pattern between those scenarios that completed, and those that didn't. It was found that for some scenarios, the distance between the start and the target was at most 10 grid points, and no terrain obstacles were in the path. These were the scenarios that completed. Of the scenarios that didn't complete, it was found that the path lengths were greater than 10 points, and there was terrain obstructing a clear path from the starting point to the target.

Analysis: At the time, it was unclear why the code had difficulty navigating a path around obstacles. One possibility was that the code did not have an explicit CLOSED list. Without an explicit CLOSED list, the code can evaluate the same paths more than once. Because of discretization bias, there exists more than one unique path with a given cost. Figure 6.4 illustrates this for a two-dimensional grid. In the continuous case, the shortest path between start S and

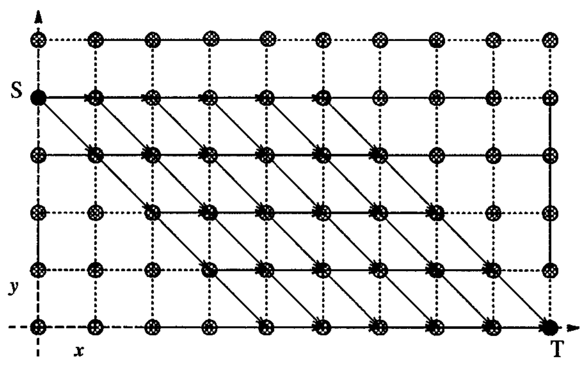

Figure 6.4 Discretization Generates Many Paths of Equal Cost from Start to Target.

target T is only one straight-line, whereas the discrete case has multiple shortest paths which are equal in discrete lengths. The greatest number of paths with non-unique costs occurs when the ratio between the x and y distances from S to T is approximately 1/2. In the figure, with $x = 9$ and $y = 4$, the number of non-unique paths is 126. In general, for $x > y$ the number of non-unique paths follows the binomial formula according to the following equation:

$$\text{Number Paths} = \frac{x!}{y!(x-y)!} \tag{6.1}$$

When $x < y$, reverse the positions of x and y in Equation 6.1 by replacing x with y and vice versa[3].

It can be seen from the equation that, as the distance between the Start and Target is increased, there can be an exponential increase in the number of paths evaluated. Without a CLOSED list to close off duplicate paths, the code may evaluate all possible paths. In addition,

[3]This relation was derived by author. Explicit enumeration of all paths was performed for small problems ($10 < x < y$). A pattern of solutions was observed which followed the binomial formula.

the OPEN list was constrained to contain a maximum of 4000 paths. When the OPEN list reached capacity, the code discarded the worst 50 paths to make room for new paths generated. This caused two problems:

1. Loss of optimality: A* is optimal only if all paths on OPEN are evaluated. Optimality is not guaranteed if paths on OPEN are discarded before being evaluated (96:p80).
2. Possibility of cycles: When paths are discarded, there exists the possibility that the same path may be put on OPEN, discarded, put on OPEN, etc., in an endless cycle.

The cause of the problems could not be determined from the code as tested with the existing terrain data files. To adequately determine the cause, additional instrumentation was needed in the code, and a better terrain model had to be developed. For instrumentation, the code was modified to record the paths and associated costs as they were put on the OPEN list. The new terrain model was developed in the shape of a geometric "S" as shown in Figure 6.5. This shape

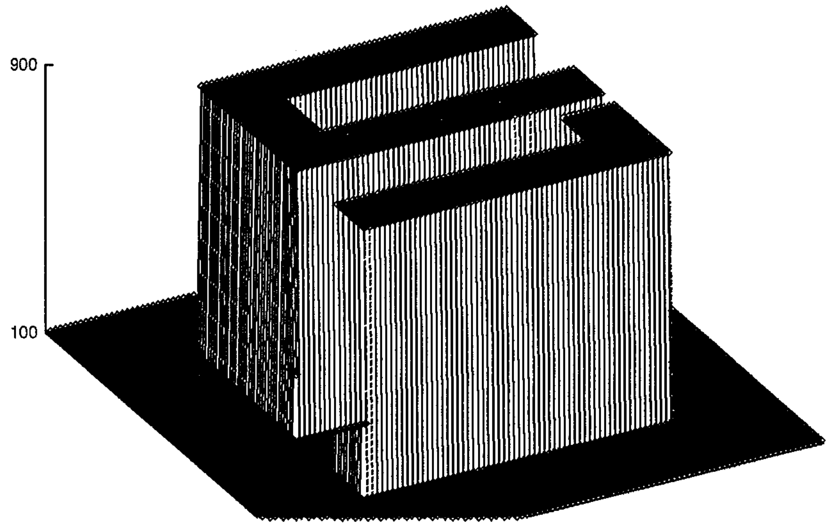

Figure 6.5 S-shaped Terrain Model Developed for This Research.

was chosen because it could test the code's ability to navigate over mountains, through and around boxed canyons. It would allow simple placement of radar to assess the code's ability to use terrain masking to avoid radar detection. It would also simplify independent validation of the optimality of a selected route. The next section describes experiments conducted with the new terrain model using old code with additional instrumentation.

6.3.2 Experiment 2: Troubleshoot Old Code. Test scenarios were developed starting from the simple (trivial) case of the Start and Target on the same grid axis with no radar and no intervening terrain. The scenarios were made more complex (*i.e.*, more realistic) by putting the Start and Target on different axes of the terrain grid, adding radar threats, and including terrain obstacles in the path. The tests were conducted on the 8-node iPSC/2 hypercube. The routes selected for three scenarios are shown in Figure 6.6. Details on the test scenarios are provided in the following sections.

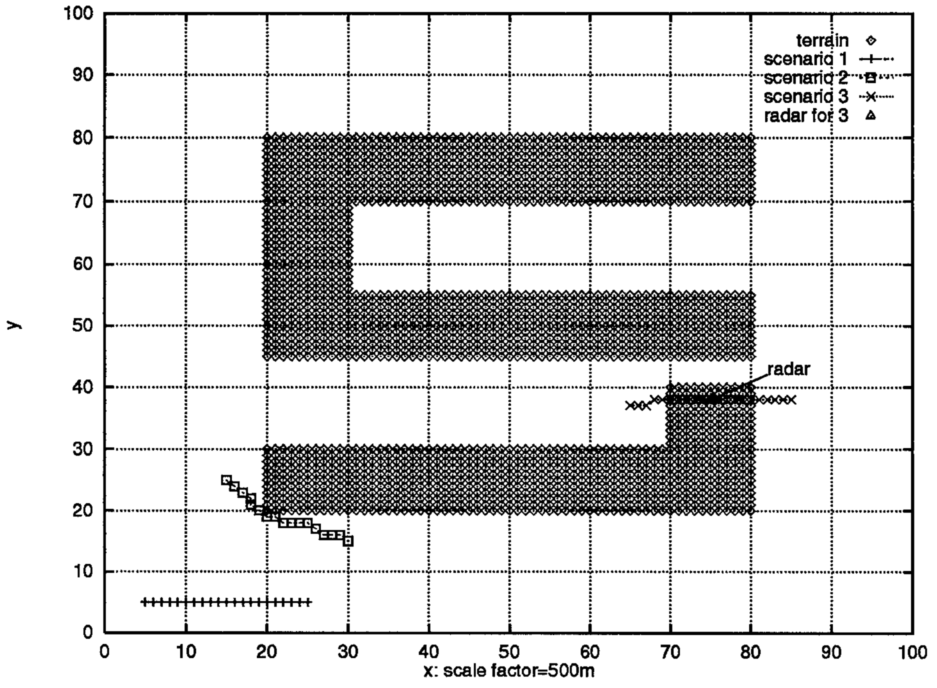

Figure 6.6 Three Routes Selected for Three Different Mission Scenarios.

6.3.2.1 Scenario 1: Path Along Axis, No Radar. In this scenario, the distance between Start and Target was 20 units on the X axis, 0 units on the Y axis, and 2 units on the

Z axis. There was no radar, and there were no terrain obstacles. An enlarged view of the selected path is shown in Figure 6.7, and the results are given in Table 6.1.

number processors	8
execution time	9.92 sec
Points in path	21
Distance cost	5.207 km
Radar cost	0.00
Paths explored	42
Paths generated	354
Paths discarded	0

Table 6.1 Results of Scenario 1.

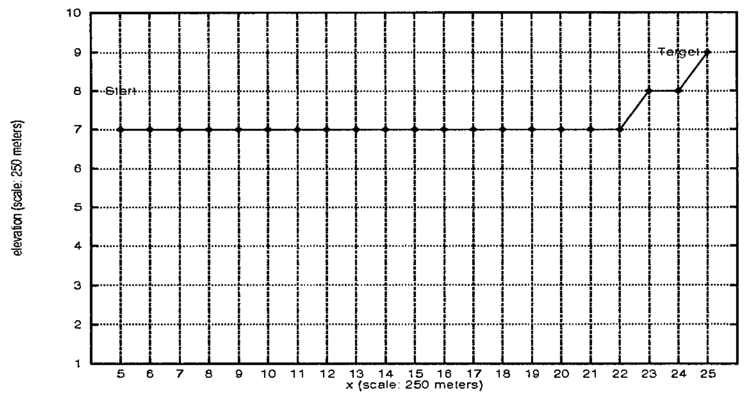

Figure 6.7 Side View of Path Plotted in Scenario 1. No change occurs on the Y axis. Path points changed only along the X and Z axes as shown.

Analysis: The code executed quickly as expected. The program calculates the straight-line distance for the heuristic. In this scenario without radar or obstacles, the heuristic function estimates the actual distance almost exactly. The scale factor per point is 250 meters. The heuristic estimate was $\sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} = 5.025$ km, which is a difference of only 1% of the computed value. The old code performs best on this type of scenario. However, this is a trivial example and not representative of real-world mission routing problems.

6.3.2.2 Scenario 2: Path Around an Obstacle, No Radar. This scenario was designed such that the code had to determine the best path by going over or around an obstacle. Radar threats were not used. The terrain obstacle was a corner of the S-shaped terrain. The terrain grid coordinates of the corner were $x, y, z = 20, 20, 4$. The results are given in Tabel 6.2, and also shown in Figure 6.8.

number processors	8
execution time	601.186 sec
Points in path	17
Distance cost	5.011 km
Radar cost	0.00
Paths explored	1704
Paths generated	10671
Paths discarded	3850

Table 6.2 Results of Scenario 2.

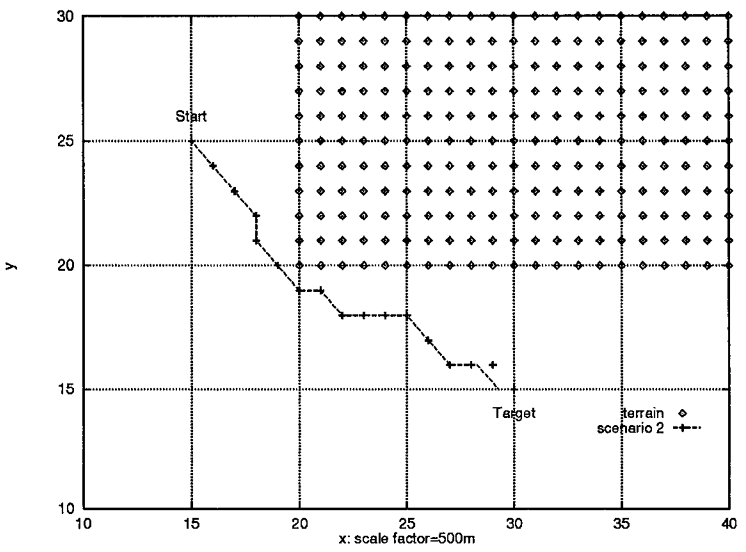

Figure 6.8 Closeup of Route Selected for Scenario 2: Path Around an Obstacle.

Analysis: Program execution for this scenario took more than 10 minutes. As the data shows, the code generated 30 times more paths for this scenario, than in scenario 1. Part of the cause for this is the discretization bias as discussed previously (see Section 6.3.1). Another cause is the terrain obstacle which forces the code to explore paths not on the direct "line-of-sight" path from Start to Target. It was not certain at this point what could be done to reduce the number of paths generated and decrease the execution time of the program. One alternative is the implementation of an explicit CLOSED list to reduce the number of paths "discarded". However, more tests were needed to further characterize the problem. The next scenario tests the code's ability to plan routes over objects and around radar.

6.3.2.3 Scenario 3: Path Over an Obstacle, With and Without Radar. This scenario was developed to determine how the code navigated over simulated mountains. One radar threat was added to assess the code's ability to avoid radar when selecting a path. The paths selected for the test cases with and without radar are shown in Figure 6.9.

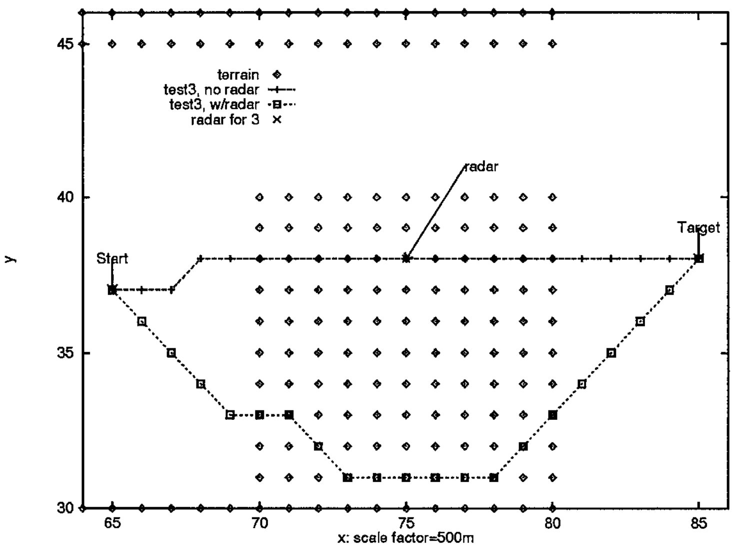

Figure 6.9 Closeup of Routes Selected for Scenario 3: Path Over an Obstacle, With and Without Radar.

Analysis: The code terminated successfully for both test cases. However, two problems were discovered:

1. The radar avoidance path was not optimal. Another path could be selected through the canyon so as to completely avoid radar.
2. It took 445 times longer to find a solution when one radar was in the direct path to the target.

The statistics are shown in Table 6.3: Adding one radar to the scenario increased the execution time by a factor of 445. A comparison between the number of paths explored, generated, and discarded offers an explanation for the disparity in execution times between the two cases. Adding one radar threat forced the code to generate and explore many more paths. This is an example of

	without radar	with radar
number processors	8	8
execution time	24.933 sec	11113.916 sec
Points in path	21	21
Distance cost	5.494 km	6.664 km
Radar cost	0.00	1016.8
Paths explored	219	23679
Paths generated	1100	138387
Paths discarded	11	94950

Table 6.3 Results of Scenario 3.

combinatorial explosion. This is the result of using a heuristic function based on only one criterion, distance, and not including the second criterion, radar cost, in the estimate. The code continually tries to minimize the distance from the current point to the target. However, radar is encountered when moving forward, so the code essentially iterates (or backtracks) around the rim of the radar coverage until a clear path to the target is discovered. This iteration causes the code to generate and regenerate paths over and over again. An examination of the OPEN list confirmed this. For the test case with radar, many duplicate paths were generated by the code and put on the OPEN list. Some paths were duplicated more than 100 times, but the average number of duplicates was approximately 50. Reducing the number of duplicates would therefore correlate directly to reducing the execution time of the program. One way of reducing the number of duplicates is by developing and using a more accurate heuristic. However, as discussed in the Section 3.6.3.1, the heuristics that include radar threat are not monotonic, take longer to calculate than simpler heuristics, and therefore may not provide a reduction in execution time. Another method which proved simple to implement and provided greater potential for reducing the number of duplicate paths was the implementation of an explicit CLOSED list. A discussion of this is given in Section 3.6.2, and the implementation details are described in Section 5.3.2. The testing of the explicit CLOSED list implementation is described in the next section (Section 6.4).

The lack of optimality in the selected path was a concern. Upon analyzing the code, two causes were discovered. One cause was that the radar weighting factor in the cost function was set too low. Because of this, the route could be exposed to radar and still be acceptable. The code would only explore those paths which were "slightly worse" in overall distance before accepting the radar path as optimal. However, it was understandable why the radar factor was set low, because a higher radar weighting factor would cause many more paths to be generated and exacerbate the problem of reducing execution time. Experiments were needed to determine the proper weighting of the radar threat to guide the search quickly to the target while avoiding radar as much as possible.

The second cause for lack of optimality was discovered in the radar model. Radar systems have a minimum detectable angle below which all objects are "hidden" from the radar. This minimum detectable angle is determined by the physical construction of the radar antenna, radar wavelength, and wave propagation effects. This angle was assigned but not implemented in the old radar model. Lack of this information prevents selection of a path that avoids radar by going under it. This would explain why the code did not select the canyon route to the target. To correct this problem, a trigonometric calculation was incorporated to determine the angle of the current position relative to the radar position. If the angle was less than the minimum detectable angle of the radar, that position was effectively hidden from the radar and would be evaluated at lower cost than one above the angle.

In the next section, details of the tests performed on the modified code are given.

6.4 Tests Performed on Modified Code

A major effort of this research concentrated on modifying and refining the original code to meet the primary objectives. The modified code included changes to the radar model to incorporate minimum detectable angle and more realistic weighting factors. Improvements to the radar model added realism to the route planner, but did not improve execution time. The greatest reduction of execution time came by adding an explicit CLOSED list. The data structure for the CLOSED list was a three-dimensional array of float values corresponding to the XYZ coordinates of the terrain gridpoints. The minimum path cost to a grid point is stored at its corresponding XYZ array location on the CLOSED list. Any time that point is encountered in a path, the cost to get to that point is compared with the cost of the path currently being evaluated. If the CLOSED cost is greater than the cost of the path currently being evaluated, the CLOSED cost is updated and the better path is put on OPEN. Otherwise, the current path is discarded. Each processor keeps an individual copy of a CLOSED list, but the information is global because each worker broadcasts the cost of each grid point that is updated on the CLOSED list. The manager process does not do any work generating paths so it does not send any CLOSED information to the workers.

The CLOSED list was implemented to record the lowest cost from the Start to each grid point encountered in the paths to the Target. A 3-D array matching the terrain grid was used to store the cost values. The array implementation requires more memory than a linked list implementation, but has faster access time ($O(1)$). The additional memory required was approximately one megabyte ($100 \times 100 \times 25$ array of 4 byte float values = 1Mbyte). This increase in memory did not impact operation of the program for the iPSC/2 or Paragon.

Communication costs for the code modification were increased because each worker broadcasts its CLOSED cost value for each path evaluated, and receives $(p-1)$ CLOSED values for p processors during each path evaluation cycle. There was concern that the added communication processing would offset any gains obtained from reducing the number of duplicates on the OPEN list.

In addition to the added communication and memory requirements for the modified code, computation time for each processor increases per evaluation cycle because of the added checks, comparisons, and assignments required for the CLOSED list implementation.

The results obtained with the modified code are discussed in the following section.

6.4.1 Results of Implementing a CLOSED List. With the explicit CLOSED list implemented in the code, the same scenario 3 was tested again. The results for this experiment are given in Table 6.4. With one radar in the path, the modified code ran significantly faster than the old code, 9 minutes vs. 3 hours! The number of paths generated, explored, and discarded was reduced by a factor of 10.

Scenario 3 test with radar		
	without CLOSED	with CLOSED
number processors	8	8
execution time	11113.916 sec	538.328 sec
Points in path	21	21
Distance cost	6.664 km	6.664 km
Radar cost	1016.8	1016.8
Paths explored	23679	2526
Paths generated	138387	14663
Paths discarded	94950	1750

Table 6.4 Results of Modified Code. Comparison of Scenario 3 test with radar, with and without a CLOSED list.

Analysis. It is clear that the number of duplicates on the OPEN list was reduced. An examination of the data recorded for the OPEN list confirmed this. In the manager/worker control structure of the old code, the worker communicated with the manager after every five path evaluations. The manager then told the worker which paths to continue exploring. With the large number of duplicates generated, the workers were spending a great amount of time doing redundant work, and redundant communication with the manager. By reducing the number of duplicates, the amount of redundant work was almost eliminated, and therefore the communication time associated with that redundant work was also reduced. The added complexity of the CLOSED list was more than offset by the large reduction in redundant work. The improvements in execution time validated the need for an explicit CLOSED list for efficient implementation of a parallel A* algorithm.

6.4.2 Termination and Load Balancing of Modified Code. The implementation of the CLOSED list yielded execution times on the order of minutes instead of hours. Some inefficiencies still existed in the modified code. One of these was the conditions for termination of the program. The termination condition for the original code depended on the distributed OPEN list at each worker processor. When the worker sent back its best points to update the global OPEN list at the manager, the points would be compared to the current global optimal solution maintained at the manager. The worker would be terminated if it sent back points that were worse than the current global optimum. This creates inefficient use of processors in the search process because one processor may be left with points to evaluate on its OPEN list, while the $P - 1$ remaining processors have terminated. This reduces the algorithm to an inefficient sequential algorithm once a path to the target has been found by one processor. In some experiments with eight processors, the last processor would terminate several minutes after the first processor that completed.

Some effort was spent trying to improve the load balancing for termination of the algorithm[4]. Various load-balancing alternatives are discussed in (75:313-315) and in (8, 9). Experiments with various approaches, such as round-robin and random polling, yielded some success in improving the termination condition. The basic approach taken is that when the goal is reached by each worker, it updates its current best solution and sends this to the manager. The manager, in turn, determines the global best solution and causes each of the workers to update its best solution with the global best. If any worker has points with lower cost than the global best, these are distributed among all the processors by the manager. The algorithm continues until no worker has points with cost less than the global best. Experiments with this method have not yielded consistent results. The code does not always find the optimal solution. There is an undetermined flaw in the old manager/worker design that allows the algorithm to terminate before the global best solution is reached. More analysis and experiments are needed to correct this problem. However, although the code did not always yield the same results on consecutive runs, the solutions obtained were "close" to each other, both in the cost of the path, and the route points of the path.

6.4.3 Speedup of Modified Code. Although inefficiencies still existed in the modified code, at this point the code was evaluated to determine what speedup could be obtained by varying the number of processors. Speedup is defined as the ratio of the serial run time of the best sequential

[4]Details are not provided in this section because after much testing it was found that this modified code did not provide any speedup over a sequential version. The discussion is provided here only to summarize the research performed for this area.

algorithm to the run time of a parallel algorithm designed to run on p processors (75). Due to the manager/worker control structure of the modified code, the minimum number of processors that could be used to test the code was two (one for the worker, and one for the manager). The first tests were performed on the iPSC/2 hypercube using the 2-processor run-time as the baseline for comparison. The same scenarios were run with 2, 4, and 8 processors. Initial results were less than promising. For one scenario, execution time improved by less than a factor of 2 when increasing the number of processors from 2 to 8. When the code was ported to the faster Paragon[5], there was even less speedup noted. In fact, some cases showed an increase in execution time as the number of processors was increased. This is due to the synchronization required between the manager and workers, as explained in the next paragraph.

The next step to get a good measure of the speedup for this modified code was to develop a version designed to run on only one processor. This sequential version would then be used as a baseline for measuring speedup. For this sequential version, all message passing was removed, and the manager was eliminated. Only the code for the worker remained. Execution times on the Paragon for this sequential code were recorded for different scenarios. In all cases, the sequential code was faster than the parallel code. How could this be? For the Paragon processors, message passing is approximately 1000 times slower than computation (milli-seconds vs micro-seconds). The sequential version is not burdened with any communication requirements, so it performs the search process unconstrained by message-passing. The parallel version of the modified code required "synchronous" message passing between the workers and the manager. Every five[6] path evaluations, the worker would send its three best paths to the manager and wait for instructions from the manager. The manager would update its global OPEN list with the inputs from the workers, and then redistribute the global best paths to the workers for further expansion. The workers were idle while waiting for the manager to send more work. In addition, the manager was idle while waiting

[5]The Paragon is a distributed memory mesh architecture of the latest i860XP RISC processors from Intel (62).
[6]The number five was parameterized so it could be changed without recompiling the code. However, varying this number did not improve the performance of the parallel code over the sequential code.

for worker inputs and did not perform any searching. This "waiting" is unproductive idle time. The manager/worker control structure was found to be inherently inefficient because one processor is dedicated to being a manager and does not contribute to the search effort except to collect and redistribute work inputs. The processors were not being efficiently used. The idle waiting time plus the communication time overcame any benefits to partitioning the problem among p processors. In addition, the amount of idle time scales with the problem size, and number of processors. As the problem size increases, the number of paths exchanged between workers and managers increases, and consequently the number of idle times also increases. Likewise, as the number of processors is increased, the number of idle times between manager and workers also increases. This corresponds with the theoretical speedup of Equation 5.4 discussed in Section 5.4.1.

The results called for the development of a new design. It was unclear what design could be implemented and tested to meet the problem requirements. Fortunately, a closer examination of the operation of the A* algorithm with the given heuristic revealed a relatively simple approach that could give performance improvements over the existing manager/worker implementation.

6.5 A New Parallel A Implementation.*

A new design had to be developed that would require less communication, more independence between processors, and elimination of the manager/worker control structure. The answer came by re-examining how the heuristic is used and implemented. The old design used the heuristic and the manager/worker control structure to maintain a global, centralized OPEN list at the manager. The manager process kept the workers focused in the global best direction as determined by the heuristic. However, the heuristic only provided an estimate of one criterion, distance remaining. The other path criterion, radar cost, was not included in the heuristic so that the heuristic would be monotonic. Because of this, the workers would always be directed towards the shortest distance paths to the target. When radar was encountered, the actual cost of path P_i to that point would be the actual distance plus the radar cost. The added cost of radar would move that path P_i lower in priority on the OPEN list and all paths of lower cost would be explored before P_i was examined again. Conceptually, this is a global best approach. It works best when the heuristic is a close approximation of the actual cost, because it keeps the search close to the optimal path. This was confirmed for scenarios which did not include radar. The distance heuristic is within 12% of the actual distance[7] and therefore guides the search quickly to a solution when no radar is in the path.

A better approach for scenarios with radar is not a global best approach, but a local best approach in a certain direction[8]. Rather than keep all processors focused in the same direction, allow each processor to independently search in a different direction. This distributed best approach would allow each processor to search for the best paths in a specific direction and perhaps discover the optimal path more quickly. This is similar to a search team which splits up to cover more ground faster than if the whole team stayed together. An added benefit to this concept is that the manager process is eliminated. Each worker maintains its own local OPEN list that is initially seeded with a path in a different direction. Intuitively, this approach showed promise for improving performance if the processors could work independently without duplication of effort. However, two questions remained:

1. How to implement the CLOSED list (updated globally or locally only), and
2. How to terminate while maintaining autonomy among worker processes.

The answers to these questions were determined by experimentation. Experiments were performed first to determine the "best" implementation of the CLOSED list, and then to develop an efficient method for algorithm termination. The next subsections discuss the tests performed to determine the final design of the new algorithm, and the test results of the new design.

[7]As discussed in Chapter 4.

[8]This is a unique approach with no direct references in published literature. The closest related work is described in (32)

6.5.1 Test 1: Local vs. Global CLOSED List. Experiments were performed to determine the best approach for implementing the CLOSED list in this design. A local CLOSED list would not require any communication among processors and would therefore give the most independence between workers. However, any overlap or duplication of effort could only be reduced with a CLOSED list that was updated with information from other workers. The experiments determined which provided better performance.

6.5.1.1 Experiment 1.1: Local CLOSED List. In the first experiment, the CLOSED list was updated only locally. Each worker process was initialized with a path in a different direction. From that point on, the worker performed the A* search algorithm independently of other workers. Each worker process terminated when the target was reached. There was no communication between processors after initialization. This approach essentially replicates the sequential code on each processor with a different initial path as the starting point. This approach was taken as a first step in developing the new design and was not expected to be efficient because no method to reduce duplication between processors was employed.

Results and Analysis for 1.1. This approach did not provide any speedup over the sequential code. Some processors terminated in slightly less time than the sequential code, but other processors ran slightly longer than the sequential code. An examination of the OPEN lists provided an explanation for this situation. There were no duplicates on an individual worker's OPEN list, but duplication was observed when the OPEN list of one worker was compared to the OPEN list of another. Although each worker started out in different directions, the heuristic guided each worker in the same direction towards the target. The result was that the search paths for each worker would overlap as the search front moved away from the starting point and towards the target. In order to reduce the overlap, the workers must communicate with each other to report the paths already explored. This requires global information to be kept in the CLOSED list on each worker processor. This global CLOSED list was implemented and tested, and the results discussed in the next section[9].

6.5.1.2 Experiment 1.2: Global CLOSED List. For this implementation, each worker maintains a copy of the CLOSED list, but it is updated with global information[10]. As a worker evaluates a path, it updates its own CLOSED list and broadcasts this information to all other workers. The workers update their own CLOSED lists with this shared information. This approach coordinates the search among the workers so there is less overlap among the individual search paths. Termination is kept the same as in Experiment 1.1: each worker terminates when it reaches the target, or when its OPEN list is empty.

Results and Analysis for 1.2. The results from this approach were very promising. With p processors, some would terminate p times faster than the sequential code for the scenarios tested. Other processors would take longer execution time than the sequential code. This was because the termination conditions were not coordinated among the workers. When a worker reached the target, the cost of this path was not shared with the other workers. Therefore the other workers continue searching until their OPEN lists are empty, or until the target is reached. To correct this, three refinements must be made to the design:

1. Storing cost of best completed path: If a worker reaches the Target along a completed path P, broadcast the cost of P to the other workers.
2. Load Balancing: If a worker has any paths on its OPEN list of lower cost than P, redistribute this work to processors with empty OPEN lists.
3. Termination: Terminate the program when all OPEN lists are empty.

The first refinement was relatively easy to implement. When the target was reached, the worker broadcast the path information to the other workers. The other workers would then continue searching along only those paths of less cost than this completed path.

[9]As a side note on the local CLOSED list implementation, it was observed that the CLOSED list was not needed because a monotonic heuristic was used, and the OPEN list was maintained locally.

[10]The duplicate pruning strategy described in (32) is similar to this CLOSED list implementation.

The second two refinements were not easily implemented because they are intertwined. How could the workers redistribute the remaining work and still know when all OPEN lists are empty? A token scheme was tested, but it was inefficient because a worker had to wait for the token to request and receive work. After much trial and error, a ring message-passing design proved to be the best implementation[11]. When a worker w_i empties its OPEN list, it requests work from its nearest logical neighbor w_{i+1}. If the neighbor w_{i+1} has paths on its OPEN list, it sends some paths to w_i; otherwise it passes the original request on to its neighbor. For p processors numbered 0 to $p-1$, w_{p-1} sends its requests to w_0 in order to form a ring. If the request from w_i goes completely around the ring of processors without being answered, all OPEN lists must be empty, and the program terminates.

With these last three refinements to the algorithm, the new design was finalized. The test results for this final design are given in the next section.

6.5.2 Test 2: Final Design. The final design eliminated the manager of the old design and had every processor doing useful work. An OPEN list was maintained locally on each processor, and it was initialized with a unique path in a specific direction. Each processor also kept a CLOSED list updated with information sent from other workers. This CLOSED list reduced duplication of effort among the workers. To keep all workers busy until termination, load-balancing was implemented in a ring message-passing design as described in the previous section. This constituted the final design. To adequately test this design, new scenarios were developed that were more realistic than the previous scenarios. Each of these scenarios involved different placements of 15 radars with the Start and Target separated by 55 points (which equates to 27.5 kilometers). In all scenarios, the new design showed close to or better than linear speedup through 16 processors on the Paragon. The results reported in this section pertain to the scenario shown in Figures 6.10, and 6.11. The code selected the route of least cost by taking advantage of the minimum detectable angle of the

[11]This is similar to Beard's load balancing strategy for the SCP problem (9).

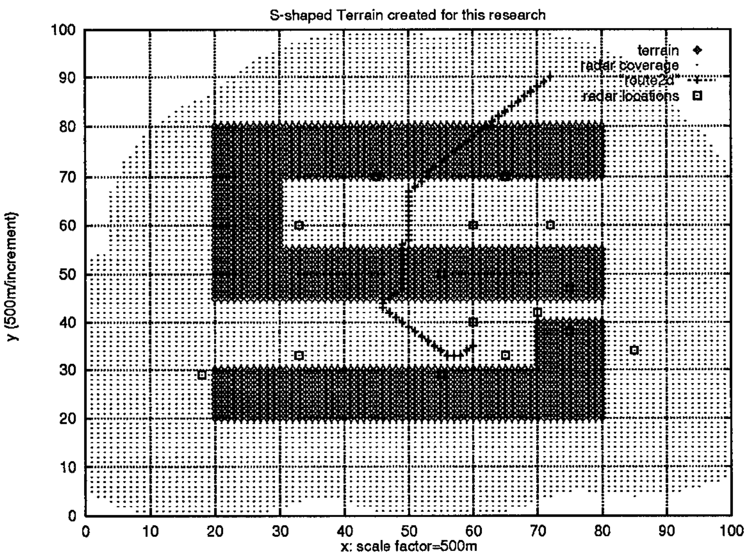

Figure 6.10 2-D view of test scenario for final A* design.

radars, and using the terrain to hide from the radars. Although it appears in Figure 6.10 that the radar covers all areas of the selected route, only 12 of 62 route points are exposed to radar. Those points and associated radar costs are shown in Table 6.5, and the statistics of the route selection are given in Table 6.6.

X	Y	Z	Radar Cost
53	70	5	2500.000000
49	55	5	1123.583618
49	54	5	931.230713
49	49	5	302.824402
49	48	5	342.773132
50	47	5	476.856323
50	46	5	377.751709
50	45	5	422.208862
50	44	4	2795.085205
50	43	3	2795.085205
59	34	2	3196.327881
60	35	3	3281.737793

Table 6.5 Data on Points Exposed to Radar from a Test of the New Code Using a Scenario With 15 Radars.

Optimality. The first question to ask is this: Did the code select the optimal path? A "yes" answer to this would meet the first objective. For scenarios with only a few radars,

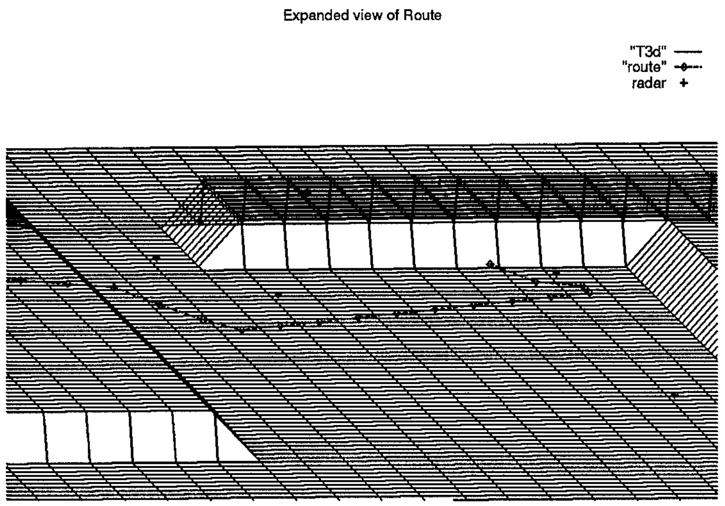

Figure 6.11 Close-up View of the Route Selected for a 15 Radar Scenario.

number processors	16
execution time	113.7 sec
Points in path	62
Distance cost hline Radar cost	37.648 km 18545.5
Paths explored	6852
Paths generated	34994
Paths discarded	153

Table 6.6 Test Results of New Code on the Paragon With a Scenario Using 15 Radars.

it is relatively simple to validate by manual calculations the optimality of a selected route. In all scenarios tested with less than four radars, the code consistently selected the same route which was manually validated to be optimal. However, with 15 radars, manual validation of the optimal route is extremely difficult. Therefore, test cases were repeated multiple times to determine if the code consistently found the same solution. For every case, the parallel code selected a route with cost equal to or lower than the sequential code. However, among multiple runs with the same number of processors, the code would sometimes find a route with lower overall cost than in previous runs. At first, the implementation of the OPEN list was suspected. The OPEN list is implemented as a priority queue, and an examination of the OPEN list implementation uncovered no apparent errors. However, for small numbers of processors ($p \leq 4$) the OPEN list would become full. The OPEN list was constrained to a fixed maximum size of 12,000 paths. The implementation was such that when OPEN became full, the 50 worst paths would be discarded to make room for newly generated paths. In one test run with the sequential code, the OPEN list became full as many as 400 times so that more than 20,000 paths were discarded. More memory per processor is needed to store all generated paths on the OPEN list for small numbers of processors[12]. This is a common complaint of the A* algorithm (109). However, for the new design developed and tested here, as the number of processors is increased, the work is distributed such that the OPEN list on each processor never fills up. This was observed in all cases using 8 or more processors.

Another possible explanation for the inconsistency among solutions may be in the implementation of the CLOSED list. In this implementation, only the lowest cost to a point is stored on the CLOSED list. However, because there is an order dependence among points, a more accurate implementation would store the lowest cost associated with reaching the point from a certain direction. This more accurate CLOSED list requires much more memory to implement because there are 26 possible directions from which to get to a point in three dimensions. To cover all directions to every one of 250,000 grid points would require as much as 26 Mbytes of additional memory for each processor. This was beyond the capacity of the Paragon system used in this research. Without further experimentation, the actual cause of the variant solutions could not be determined. However, an examination of the difference in cost between routes selected on multiple runs revealed a cost variance of less than 3%. In addition, the routes selected differed by only a few points between them. Further experimentation in this area was left for future research.

Execution Time. The second objective of the MRP problem was to reduce the execution time to 4.5 minutes or less for the simplified MRP problem used in this research. Figure 6.12 reports the execution time for one scenario with 15 radars. With one processor, the

[12]In the current array implementation of OPEN, 14.2 Mbytes are required to store 12,000 paths. Experiments were not performed to determine exactly how much more memory is needed to prevent OPEN from ever becoming full.

route was selected in approximately 27 minutes. With 16 processors, that time was reduced to less than two minutes.

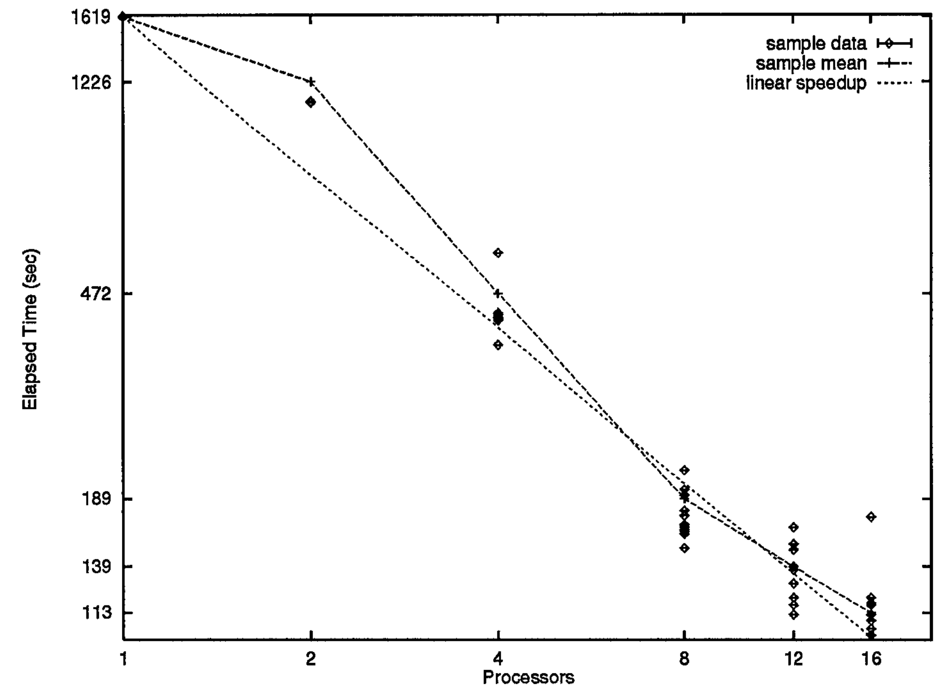

Figure 6.12 Execution Time Versus Number of Processors for a Scenario With 15 Radars.

Figure 6.13 reports the same information in terms of speedup. Notice that for 8 to 12 processors, the speedup is greater than linear. This is sometimes referred to as superlinear speedup (37, 99), and it was also observed by Beard in his parallel SCP[13] algorithm. Although some consider superlinear speedup impossible to achieve (35), others support it (99:p81), and Kumar acknowledges that acceleration anomalies (*i.e.* superlinear speedup) may be observed in parallel search algorithms (75:p337). This happens when one of the p processors discovers the optimal path quickly and reduces the number of paths explored over the sequential algorithm. The parallel algorithm essentially performs less searching than the sequential algorithm. This produces superlinear speedup in the new parallel A* design when compared against the sequential A* algorithm[14].

[13]SCP = Set Covering Problem.

[14]Some would argue that the sequential algorithm used for comparison is not the fastest. This may be true, and is discussed in Section 7.2.2.1.

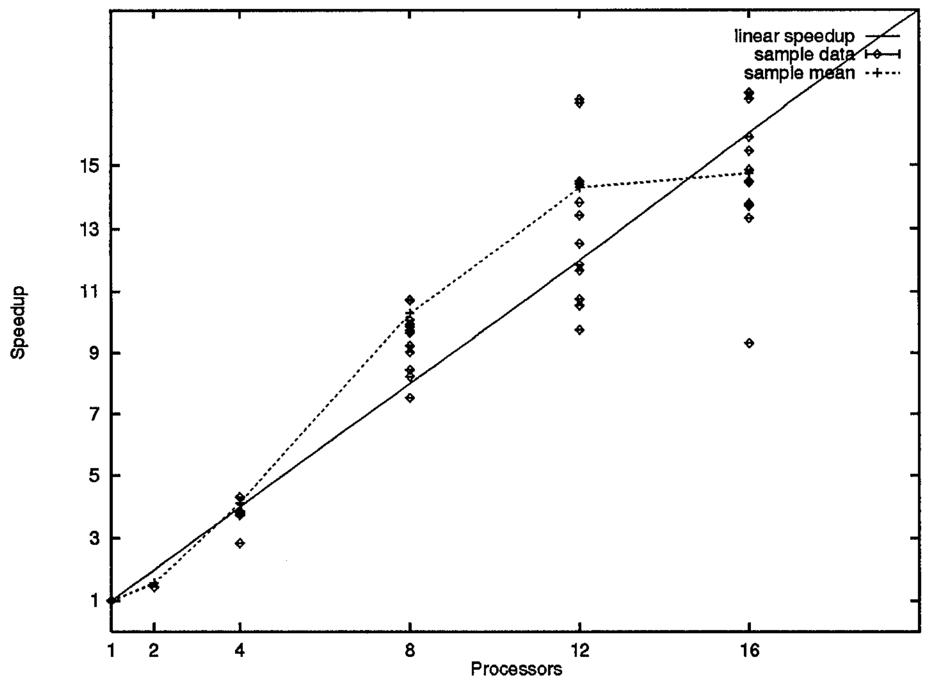

Figure 6.13 Measured Speedup for Route Calculations With 15 Radar Threats.

In general, this new design yielded speedup values close to linear through 16 processors on the Paragon. This satisfies the second objective for the scenarios tested. For the first objective, optimality of the solution could not be guaranteed. However, experiments showed variances of less than 3% between routes selected by multiple runs of the parallel code. In addition, the parallel design distributed work such that the OPEN list on each processor never became full when 8 or more processors were used. This improved optimality over the sequential version.

6.6 Summary

This chapter documented the experiments conducted on different versions of code to assess how well the primary objectives were met. Initial experiments uncovered serious flaws in the old algorithm design reported in (29). The lack of a CLOSED list prevented the code from terminating on all but the most trivial scenarios. To correct this, the code was modified to incorporate a CLOSED list, and a new S-shaped terrain model was developed to facilitate testing for proper operation of the code. The CLOSED list reduced the number of duplicate work performed by the processors. Orders of magnitude improvement in execution times were noted with the explicit CLOSED list version of the code. These improvements in the execution time allowed larger size scenarios to be tested. However, it was discovered that although the modified code was significantly faster than the original code for large problems, there was no appreciable improvement in execution time when the number of processors was increased. Therefore, to test for speedup, a sequential version of the algorithm was developed and tested against the parallel version. The results showed that for all experiments on the old design, the sequential version ran faster than the parallel version. The manager/worker structure of the old design required synchronization that caused unproductive idle periods in the search process. This prompted development of a new parallel version of the A* algorithm. The old manager/worker control structure was abandoned in favor of nearly autonomous workers. Each processor worked from its own local OPEN list, but kept a copy of a global CLOSED list. The new implementation performed beyond expectations. Experiments recorded linear or better speedup through 16 processors. Speedup leveled off with more than 16 processors unless the problem size was increased. According to the scalability analysis given in Section 5.4.1, the linear performance may be achieved on larger numbers of processors with appropriate scaling of the problem size. However, for the scenarios tested in this research, 16 processors provided real-time speed. In one experiment, with 16 processors the new program selected an optimal route of 116 gridpoints through 15 radars in less than two minutes. In all experiments, the code finished in less than 4.5 minutes with 16 processors. Further refinements to the code may improve this performance and bring us "closer" to a real-time operational mission route planner.

VII. Conclusions and Recommendations

7.1 Introduction

Timely planning of aircraft mission routes is essential to success on the battlefield. The goal of this research effort is to determine the feasibility of an effective and efficient method for automating the mission route-planning function. The previous chapter presents the results of experiments performed to validate the performance of the design. In this chapter, conclusions are drawn from the results of the experiments, and recommendations for future research are presented.

7.2 Conclusions

For this research effort, the general multicriteria MRP problem is simplified to planning a route for one aircraft to one target using only two criteria (distance and radar exposure). Experiments validated the feasibility of this simplified MRP problem. Optimal solutions are obtained for scenarios with 15 radars with execution times of less than ten minutes using 16 parallel processors on the Intel Paragon. This research extended the previous efforts of Grimm (50) and Droddy (29), and laid the groundwork for application to the general MRP problem. From this research, the following general conclusions can be made:

- The A* search is streamlined (made more efficient) by including an explicit CLOSED list and by eliminating the Manager/Worker control structure of the parallel algorithm.
- Execution time is greatly improved to make real-time route-planning closer to reality.
- Judicious design of the experiments greatly facilitates performance evaluation, including proof of optimality.
- Greater realism was introduced by improving radar cross-section estimation and the simultaneous handling of minimum distance and minimal radar exposure.

The specific conclusions drawn from the research are described in the following subsections:

7.2.1 Problem Complexity. Although the MRP problem was shown to be NP-Complete (see Section 2.5), a search graph transformation can be made so that shortest-path solution techniques with $O(n^2)$ complexity can be used (See Section 2.4). The A* algorithm approach selected for this research reduces the number of n that must be searched explicitly, so that the time complexity is $O(l^2) \leq O(n^2)$, where l is the number of points in the optimal path.

7.2.2 Algorithm. The A* algorithm was selected over other algorithms because with proper choice of the heuristic, A* dominates (*i.e.* performs as well as, or better) than all other algorithms with access to the same heuristic (96:p85). To efficiently parallelize the A* algorithm, a thorough understanding of the problem is required. For the bicriteria MRP problem, a "local best" approach was found to be more efficient than a "global best" approach. The implementation developed for this "local best" approach eliminated the interprocessor synchronization required by the manager/worker "global best" approach used in previous research (29, 50). In addition, a global CLOSED list was found to be essential to reducing duplicate work and thereby improving efficiency.

7.2.2.1 Speedup. The results of the speedup experiments showed that for some scenarios the new parallel A* algorithm had superlinear speedup over the sequential A* algorithm. For this to occur, the parallel algorithm must be doing less work than the sequential algorithm. This indicates that the design of the sequential algorithm can be improved. The improvement in execution time for the parallel algorithm is due to the use of multiple distributed OPEN lists, with duplicates eliminated using a global CLOSED list[1]. The same approach can be duplicated in a sequential algorithm two ways: multiple unique OPEN lists and a single CLOSED list can be implemented in one sequential program, or multiple tasks each having a single OPEN and CLOSED list can be implemented to run on a single processor. This proposed design for a sequential algorithm would mimic the work done by the parallel algorithm and would be a better comparison for speedup experiments.

[1]A copy of the CLOSED list is stored at each processor and kept globally current through interprocessor message passing.

7.2.2.2 Scalability. The algorithm scaled well on the Paragon with tests of up to 20 processors. Near linear speedup, or better, was obtained for all test cases (see Sections 5.4.1 and 7.2.2.1 for discussion). For larger numbers of processors, it was found that the problem size must be increased to achieve the same level of performance. This indicates that the algorithm may perform well with larger problems which are representative of real mission scenarios.

7.2.3 Terrain Model. The grid structure used for the terrain model was simple to implement and effective in providing dynamic calculations of radar exposure. The array implementation provided quick $O(1)$ access to the information without a large burden on memory (39 KB for a 100×100 surface grid).

7.2.4 Radar Model. Dynamic calculations for radar exposure is simpler to implement than static radar calculations used in (50), and requires very little memory to implement (less than 1 KB). The dynamic approach does not need any pre-processing; it performs radar calculations only as needed by the program. It provides the capability for using multiple radar cross-sections to plan routes for "low-observable" aircraft. Also, updating radar information is performed simply by updating the radar file, rather than recalculating an extensive matrix of radar threat data. Consequently, the dynamic approach is the most promising approach for a real-time mission route planner.

7.3 Recommendations for Future Research

This research continues to lay the groundwork for developing a real-time mission route planner, but it is by no means complete. There are several research projects that would bring us closer to an operational implementation. Some general research questions are

- Is there a theoretical approach to proving optimality, rather than relying on empirical methods?
- Is there a formal relationship between A* and dynamic programming algorithms? Does A* dominate dynamic programming?

Specific topics for future research fall into four main categories:

1. Algorithm improvements.
2. More realistic models.
3. Additional route criteria.
4. Graphics and visualization

These are discussed in the following subsections:

7.3.1 Algorithm Improvements. Several possibilities exist for the improvement of the current algorithm design. One area of improvement is in the implementation of the OPEN list. It is currently implemented as a priority queue array structure with a serial ordering of paths. The delete operation takes only $O(1)$ time, but the insert operation can take $O(m)$ time for m paths in the queue. With up to 12,000 paths in the queue, the insert operation can be a significant contributor to the execution time. A heap may provide a more efficient implementation of OPEN. It would perform both the insert and delete operations in $O(\log m)$, so it may give better performance over the current implementation, depending on the number of delete operations are performed compared to the number of insert operations.

Another area of improvement would be in the updating of the CLOSED list. Currently, each processor broadcasts to all other processors each local path that is closed. This allows each processor to maintain global information in its local copy of the CLOSED list. However, as the number of processors is increased, this broadcasting increases the communication burden and reduces the amount of speedup that can be obtained for larger numbers of processors. As discussed in Section 5.4.1, the processor communication required for the CLOSED list contributes $O(\sqrt{p})$ time per processor for each grid point evaluated. This could become significant for large numbers of processors p. This was not seen through 20 processors on the Paragon, but may become more apparent as the problem is scaled through larger numbers of processors. Additional research in this area would focus on evaluating and validating the scalability of the current design and determining improvements if needed.

A third area of improvement pertains to the sequential A* algorithm. The general approach calls for only one OPEN list, but this research indicated that multiple unique OPEN lists may be more efficient for multicriteria problems. Various ways to implement multiple OPEN lists in a sequential A* algorithm design should be explored. Research in this area may in turn provide more insight into the parallel design and lead to improvements in the parallel efficiency.

In addition to those recommended improvements listed above, another topic from the operations research community is to modify the algorithm to produce a set of non-dominated (optimal) solutions, rather than a single optimal solution. This would provide the user of the MRP system with a number of different routes that have approximately equal cost.

7.3.2 More Realistic Models. Improvements can be made to the models used in this research to provide a closer representation to real-world scenarios. The models for the terrain, radar, and aircraft movement can be improved, and the effects on the time performance and storage space requirements for these improvements should be determined.

Terrain: The implementation for the terrain should be upgraded to provide the capability for actual terrain data obtained from the Defense Mapping Agency (DMA). The current terrain model uses a "flat earth" model with artificially generated terrain to simulate real-world terrain. The model should be upgraded to incorporate a spherical earch representation that is provided by DMA. In addition to these terrain model improvements, the feasibility of mapping edges to points in the search graph should be explored. It was determined in Section 2.4 that this type of mapping could be done to apply shortest-path algorithms to the MRP problem to produce paths with optimal substructures. Research should explore ways to implement this and determine the effects on performance.

Radar: The calculations for probability of radar detection have been simplified for ease of implementation. More realism can be added by including probability of failure, varying the receiver noise figures, and including more precise calculations for both monostatic and bistatic radar calculations. In addition, multiple RCSs should be incorporated. For example, a front, side, and rear view RCS can be used so that the route planner can plan better routes for low-observable aircraft.

Aircraft: The current model limits aircraft turns, climbs, and descents to 45° or less, based on discrete movement from one point in the grid to an adjacent point in the grid. This proved adequate for this research, but a more accurate model would allow aircraft movement based on the performance of the aircraft. The aerodynamics of aircraft movement could be incorporated to allow more accurate turns and greater precision in aircraft orientation. This, in turn, would improve the precision of the radar calculations.

7.3.3 Additional Route Criteria. Future research could include additional facotrs for the path selection and associated complexity analysis. Examples of additional path evaluation factors include:

- time constraints
- fuel constraints
- multiple aircraft speeds
- weather
- multiple targets

These are all important factors (criteria and constraints) which are evaluated in real-world mission route-planning, so including them would make the system more realistic and more usable. Determining ways to implement these criteria and their effects on the complexity of the problem is important in contributing to the goal of an operational MRP system.

7.3.4 Graphics and Visualization. The current implementation is command line driven, and uses data files for the inputs. The program outputs data in text file format which must be interpreted for the results. The program interface should be improved so that the results could be more easily interpreted. Provide a "point-and-click" mouse system with pull-down menus to quickly update or change radar and aircraft characteristics. Add graphics visualization so that when the route is selected, a pilot can "fly" the route over the terrain. This is called simulated mission rehearsal (1, 2). These graphics improvements would make the system simpler to use, and would be enhancements requested by the user (52).

7.4 Summary

The goal of this research investigation is to determine the feasibility of a real-time optimal mission route planner. Specifically, the research goal is to design an automated system with two primary objectives:

1. Select an optimal route from base to target through defended terrain.
2. Find the optimal route in 4.5 minutes or less for the simplified bicriteria MRP problem.

The objectives are achieved and demonstrated for the simplified MRP problem of planning a route for one aircraft to one target with two criteria (distance and radar exposure) for the route. Problem scenarios with 15 radars are tested. The approach uses a parallel A* algorithm implemented on a Paragon mesh parallel architecture. The OPEN lists are distributed, with duplicates pruned by a global CLOSED list strategy. The implementation is fast and efficient on the Paragon, and scales well through 20 processors. A side benefit of this strategy is that it points out improvements for the sequential A* algorithm and it shows applicability to similar types of multicriteria optimization problems.

This research continues the previous work of Grimm (50) and Droddy (29) in laying the groundwork for extendion to the general MRP problem. Many avenues of additional research can be explored from this point, depending on the area of expertise and the interests of the next researchers. The ultimate goal of this research is a real-time operational system that may be used on the ground and in the aircraft to provide fast mission route-planning. This research brings us another step closer to realizing that goal.

Bibliography

1. Advisory Group for Aerospace Research & Development (AGARD). *Mission Planning Systems for Tactical Aircraft (Pre-Flight and In-Flight).* Technical Report AGARD-AR-296, North Atlantic Treaty Organization (NATO), May 1991. DTIC Number: AD-A237 855.
2. Advisory Group for Aerospace Research & Development (AGARD). *Mission Planning Systems for Tactical Aircraft (Pre-Flight and In-Flight).* Technical Report AGARD-AR-313, North Atlantic Treaty Organization (NATO), December 1992. DTIC Number: AD-A264 176.
3. Aho, Alfred V., et al. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.
4. Austin, Steven. "An Introduction to Genetic Algorithms," *AI Expert* (March 1990).
5. Baase, Sara. *Computer Algorithms: Introduction to Design and Analysis* (2nd Edition). Addison-Wesley, 1988.
6. Bahnij, Maj Robert B. *A Fighter Pilot's Intelligent Aide for Tactical Mission Planning.* MS thesis, AFIT/GCS/ENG/85D-1, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1985.
7. Barbehenn, Michael and Seth Hutchinson. "Efficient Search and Hierarchical Motion Planning Using Single-Source Shortest Paths Trees." *IEEE International Conference on Robotics and Automation,1*. 566–571. 1993.
8. Beard, Ralph A. *Determination of Algorithm Parallelism in NP Complete Problems for Distributed Architectures.* MS thesis, AFIT/GE/ENG/90D, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1990.
9. Beard, Ralph A. "Determination of Algorithm Parallelism in NP Complete Problems for Distributed Architectures." *Fifth Distributed Memory Computing Conference,1*. 42–51. April 1990.
10. Bogetoft, Peter and Peter Pruzan. *Planning with Multiple Criteria.* Elsevier Science Publishers B.V., 1991.
11. Bonds, Ray, editor. *The Great Book of Modern Warplanes.* Salamander Books Ltd., 1987.
12. Bradshaw, 2Lt Jeffrey S. *A Pilot's Planning Aid for Route Selection and Threat Analysis in a Tactical Environment.* MS thesis, AFIT/GCS/ENG/86D-11, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1986.
13. Bramanti-Gregor, Anna. *Strengthening Heuristic Knowledge in A* Search.* PhD dissertation, Wright-State University, 1993.
14. Brassard, Gilles and Paul Bratley. *Algorithmics: Theory and Practice.* Prentice Hall, 1988.
15. Bultan, Tevfik and Cevdet Aykanat. "A Mean Field Annealing Algorithm for the Mapping Problem." *Intel Technology Focus Conference.* 273–308. April 1992.
16. Canny, J. *The Complexity of Robot Motion Planning.* MIT press, Cambridge, MA, 1988.
17. Canny, J. and J. H. Reif. "New Lower Bound Techniques for Robot Motion Planning Problems." *28th Annual IEEE Symposium on the Foundation of Computer Science.* 49–60. 1987.
18. Carroll, K. P., et al. "AUV Path Planning: An A* Approach to Path Planning with consideration of Variable Vehicle Speed and Multiple, Overlapping, Time-Dependent Exclusion Zones." *IEEE Symposium on Autonomous Underwater Vehicle Technology.* 79–84. 1992.

19. Chan, Yupo. "Facility Location and Land Use: Multicriteria Decision Making Procedures." Forthcoming Textbook, August 1994.

20. Chandy, K. Mani and Jayadev Misra. *Parallel Program Design: A Foundation.* Addison Wesley Reading, MA, 1988.

21. Cheeseman, Peter, et al. "Where the *Really* Hard Problems Are," *Automated Reasoning*, 331–337 (1992?).

22. Chen, Y. L. and Y. H. Chin. "The Quickest Path Problem," *Computers and Operations Research*, *17*(2):153–161 (1990).

23. Cheng, Linfu and John D McKendrick. "Autonomous Knowledge-Based Navigation in an Unknown Two-Dimensional Environment with Convex Polygon Obstacles.." *Proceedings of SPIE Applications of AI VII1095*. Bellingham, WA: SPIE, 1989.

24. Christofides, Nicos. *Graph Theory: An Algorithmic Approach.* Academic Press, 1975.

25. Climaco, J. and E. Martins. "A bicriterion shortest path algorithm," *European Journal of Operational Research*, *11*:399–404 (1982).

26. Cormen, Thomas H., et al. *Introduction to Algorithms.* McGraw Hill, 1992.

27. Corrigan, J.D. and K.J. Keller. "Pilot's Associate: An Infilght Mission Planning Application." *AIAA Guidance, Navigation, and Control Conference.* August 1989.

28. Donald, Bruce, et al. "Kinodynamic Motion Planning," *Journal of the ACM*, *40*(5):1048–1066 (November 1993).

29. Droddy, Vincent A. *Multicriteria Mission Route Planning Using Parallelized A* Search.* MS thesis, AFIT/GCE/ENG/93D-04, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1993.

30. Droddy, Vincent A. and others. *Compendium of Parallel Programs for the Intel iPSC Computers.* Technical Report, Wright-Patterson AFB, OH: Air Force Institute of Technology, 1993.

31. Droddy, Vincent A., et al. "Parallel Real-Time Multicriteria Mission Routing." *Intel Supercomputer Users Group.* 259–270. October 1993.

32. Dutt, Shantanu and Nihar R. Mahapatra. "Parallel A* Algorithms and their Performance on Hypercube Multiprocessors." *Seventh International Parallel Processing Symposium.* April 1993.

33. Dyer, Douglas E. and Gregg H. Gunsch. "Low-Level Aerial Route Planning for Detection Avoidance." Unpublished, June 1992.

34. (ESC), Electronic Systems Center. "CLOAR System Requirements Document." Contract F33657-93-C-2130, Counter Low Observable Auto Router (CLOAR), July 1993.

35. Faber, V., et al. "Superlinear speedup of an efficient sequential algorithm is not possible," *Parallel Computing*, *3*:259–260 (1986).

36. Fan, Kuo Chin and Po Chang Lui. "A Path Plannning Algorithm for a Mobile Robot.." *Proceedings of SPIE Applications of AI IX 1468*. Bellingham, WA: SPIE, 1991.

37. Fischer, Dietrich. "On superlinear speedup," *Parallel Computing*, *17*:695–697 (September 1991).

38. Flood, Merrill M. "The Travelling Salesman Problem," *Journal of the Operations Research Society of America*, *4*:61–75 (1956).

39. Frakes, William B., et al. *Software Engineering in the Unix/C Environment*. Prentice-Hall, Inc., 1991.

40. Garey, M. R. and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

41. Garfinkel, Robert S. and George L. Nemhauser. *Integer Programming*. John Wiley & Sons, 1972.

42. Garmon, Joel S. *Implementation and Analysis of NP-Complete Algorithms on a Distributed Memory Computer*. MS thesis, AFIT/GE/ENG/92-M, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, March 1992.

43. Glassner, Andrew S. *Graphics Gems*. Academic Press, 1990.

44. Goldberg, David E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

45. Golden, Bruce L., et al. "The Orienteering Problem," *Naval Research Logistics*, *34*:307–318 (1987).

46. Golden, Bruce L., et al. "A Multifaceted Heuristic for the Orienteering Problem," *Naval Research Logistics*, *35*:359–366 (1988).

47. Goldsmith, Jeffrey and John Salmon. "Automatic Creation of Object Hierarchies for Ray Tracing," *Computer Graphics and Applications*, 14–20 (May 1987).

48. Golshani, F., et al. "RUTA-100: A Dynamic Route Planner." *Proceedings of the IEEE International Conference on Tools with AI*. 418–423. 1992.

49. Grama, Ananth Y. and Vipin Kumar, "Parallel Processing of Discrete Optimization Problems: A Survey," November 1992. University of Minnesota.

50. Grimm, James J. *Solution to a Multicriteria Aircraft Routing Problem Utilizing Parallel Search Techniques*. MS thesis, AFIT/GCE/ENG/92D-04, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1992.

51. Grimm, James J., et al. "A Parallelized Search Strategy for Solving a Multicriteria Aircraft Routing Problem." *Proceedings of the 1993 ACM/SIGAPP Symposium on Applied Computing*. New York: ACM Press, February 1993.

52. Gudaitis, Capt James, "Personal Interview on Mission Route Planning," November 1993. Capt Gudaitis flew in DESERT STORM as a navigator on FB-111 aircraft.

53. Hanley, Arthur. "Automated Mission Planning: A Striking Capability," *Aerospace America*, *30*(3):34–38 (March 1992).

54. Harel, David. *Algorithmics: The Spirit of Computing*. Addison-Wesley Publishers Limited, 1987.

55. Hart, P. E., et al. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths.," *IEEE Trans. Systems Science and Cybernteics*, *4*(2) (1968).

56. Herman, Martin. "Fast, Three-Dimensional, Collision-Free Motion Planning." *IEEE International Conference on Robotics and Automation2*. 1056–1063. 1986.

57. Hewish, Mark and Gérard Turbé. "A Multifaceted Heuristic for the Orienteering Problem," *International Defense Review*, *24*(7, suppl.):85–90 (July 1988).

58. Hung, Yung-Chen and Gen-Huey Chen. "Distributed algorithms for the quickest path problem," *Parallel Computing*, *18*(7):823–834 (July 1992).

59. Hura, Myron and Gary McLeod. *Route Planning for Low Observable Aircraft and Cruise Missiles.* Technical Report MR-187-AF, RAND Corp., 1993.

60. Hyland, John C. and Stan R. Fox. "A Comparison of Two Obstacle Avoidance Path Planners for Autonomous Underwater Vehicles." *IEEE Symposium on Autonomous Underwater Vehicle Technology*. 1990.

61. "Intel Corporation", "iPSC/860 and IPSC/2 C System Calls Reference Manual," March 1992. rev 001.

62. "Intel Corporation", "Intel Paragon Supercomputers," October 1993. Technical Brochure; Order No. 203/003 10-93/5K/JP.

63. "Intel Corporation", "Paragon C System Calls Reference Manual," June 1994.

64. Isensee, Ernst K. *Multicriteria Network Routing of Tactical Aircraft in a Threat Radar Environment*. MS thesis, AFIT/GST/ENS/91-M-01, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, March 1991.

65. Jacobs, Paul and John Canny. "Planning Smooth Paths for Mobile Robots." *IEEE International Conference on Robotics and Automation1*. 2–7. 1989.

66. Johnson, Rubin, "Desktop Analysis and Routing Tool, Phase I Kickoff," March 1993. Operations Research Concepts Applied.

67. Johnson, Rubin, "Desktop Analysis and Routing Tool, Phase II Interim Report," October 1993. Operations Research Concepts Applied.

68. Keirsey, David M., et al. "Multilevel Path Planning for Autonomous Vehicles." *Proceedings of SPIE Applications of AI485*. Bellingham, WA: SPIE, May 1984.

69. Keller, C. Peter. "Algorithms to solve the orienteering problem: A comparison.," *European Journal of Operational Research*, *41*:224–231 (July 1989).

70. Kirkpatrick, S., et al. "Optimization by Simulated Annealing," *Science*, *220*:671–680 (1983).

71. Kline, Kevin Brian. *Generation of Flight Paths Using Hierarchical Planning.* MS thesis, Wright State Universtiy, 1985.

72. Korf, Richard E. "Depth-First Iterative Deepening: An Optimal Admissable Tree Search," *Artificial Intelligence*, 97–109 (1985).

73. Korf, Richard E. "Linear-space best-first search," *Artificial Intelligence*, *62*:41–78 (1993).

74. Kronsjo, Lydia. *Computational Complexity of Sequential and Parallel Algorithms.* John Wiley & Sons, Ltd., Great Britain, 1985.

75. Kumar, V., et al. *Introduction to Parallel Computing*. Benjamin/Cummings, 1994.

76. Kwak, S. H., et al. "A Mission Planning Expert System for an Autonomous Underwater Vehicle." *IEEE Symposium on Autonomous Underwater Vehicle Technology*. 123–128. 1990.

77. Lamont, Gary B., et al. *Compendium of NP-Complete Problem Analysis and Synthesis.* version 1.3 vol 1, Air Force Institute of Technology, February 1992.

78. Laporte, Gilbert and Silvano Martello. "The Selective Travelling Salesman Problem," *Discrete Applied Mathematics*, *26*:193–207 (March 1990).

79. Lawler, E. L., et al., editors. *The Traveling Salesman Problem.* John Wiley & Sons, 1985.

80. Lizza, Carl Steven. *Generation of Flight Paths Using Heuristic Search.* MS thesis, Wright State University, 1984.

81. Lyons, Gary and Diane J. Cook. "A SIMD Approach to IDA*." *SPIE Applications of Artificial Intelligence X: Knowledge-Based Systems1707*. 126–136. 1992.

82. Male, Herb, et al. "An Integrated On-Board Route Planner for Helicopter Applications." *IEEE/AIAA 11th Digital Avionics Systems Conference*. 193–197. 1992.

83. Martins, E. "On a multicriteria shortest path problem," *European Journal of Operational Research*, *16*:236–245 (1984).

84. Metea, Mark B. and Jeffrey J.-P. Tsai. "Route Planning for Intelligent Autonomous Land Vehicles using Hierarchical Terrain Representation." *Proceedings of the International Conference on Robotics and Automation*. 1947–1952. July 1987.

85. Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1992.

86. Mitchell, Joseph S. B. "An Algorithmic Approach to Some Problems in Terrain Navigation," *Artificial Intelligence*, *37*:171–201 (1988).

87. Mitchell, Joseph S. B. "L_1 Shortest Paths Among Polygonal Obstacles in the Plane," *Algorithmica*, *8*(1):55–88 (1992).

88. Mitchell, Joseph S. B. and David M. Keirsey. "Planning Strategic Paths Through Variable Terrain Data." *Proceedings of SPIE Applications of AI485*. Bellingham, WA: SPIE, May 1984.

89. Mitchell, Joseph S.B. "Algorithmic Approaches to Optimal Route Planning." *SPIE Mobile Robots V1388*. 248–259. 1990.

90. Mohan, Joseph. "Experience with two parallel programs solving the traveling salesman problem.." *Proceedings of the International Conference on Parallel Processing*. 1983.

91. Mote, John, et al. "A parametric approach to solving bicriterion shortest path problems," *European Journal of Operational Research*, *53*:81–92 (July 1991).

92. Nilsson, Nils J. *Principles of Artificial Intelligence*. Tioga Publishing Co., 1980.

93. Olsan, James B. *Genetic Algorithms Applied to a Mission Routing Problem*. MS thesis, AFIT/GCE/ENG/93D-04, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1993.

94. Pal, Prabir K. and K. Jayarajan. "Fast Path Planning for Robot Manipulators Using Spatial Relations in the Configuration Space." *IEEE International Conference on Robotics and Automation2*. 668–673. 1993.

95. Papadimitriou, C. H. and K. Stieglitz. "Some Examples of Difficult Traveling Salesman Problems," *Operations Research*, *26*(3):434–443 (1978).

96. Pearl, Judea. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

97. Pellazar, Miles B. "Multi-Vehicle Route Planning with Constraints Using Genetic Algorithms." *National Aerospace and Electronics Conference*. May 1994.

98. Piatko, Christine D. *Geometric Bicriteria Optimal Path Problems*. PhD dissertation, Cornell University, August 1993.

99. Quinn, Michael J. *Parallel Computing*. McGraw-Hill, Inc., 1994.

100. Ramesh, R. and Kathleen M. Brown. "An Efficient Four-Phase Heuristic for the Generalized Orienteering Problem," *Computers Ops Res.*, *18*(2):151–165 (1991).

101. Rich, Elaine and Kevin Knight. *Artificial Intelligence* (2nd Edition). McGraw Hill, Inc., 1991.

102. Richbourg, R. F. and Neil C. Rowe Michael J. Zyda. "Exploiting Capability Constraints to Solve Two-Dimensional Path Planning Problems." *IEEE International Conference on Robotics and Automation,1*. 90–95. 1986.

103. Rosen, J. B., et al. "Algorithms for the Quickest Path Problem and the Enumeration of Quickest Paths," *Computers and Operations Research*, *21*(2):113–118 (1991).

104. Rumbaugh, James, et al. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

105. Schulmeyer, G.Gordon and James I. McManus, editors. *Total Quality Management for Software*. Van Nostrand Rienhold, 1992.

106. Skolnik, Merrill I., editor. *Radar Handbook*. McGraw Hill, 1990.

107. Stewart, Bradley S and III Chelsea C White. "Multiobjective A*," *Journal of the ACM*, *38*(4):775–814 (October 1991).

108. Stiles, Peter and Ira Glickstein. "Route Planning." *Proceedings IEEE/AIAA 10th Digital Avionics Systems Conference*. 420–425. 1991.

109. Stiles, P.N. and I.S. Glickstein. "Highly Parallelizable Route Planner Based On Cellular Automata Algorithms," *IBM Journal of Research and Development*, *38*(2):167–181 (March 1994).

110. Teng, Y. Ansel, et al. "Stealth Terrain Navigation," *IEEE Transactions on Systems, Man, and Cybernetics*, *23*(1):96–109 (January 1993).

111. Thangiah, Sam R. and Kendall E. Nygard. "Dynamic Trajectory Routing using an Adaptive Search Method," *ACM/SIGAPP Symposium on Applied Computing*, 131–138 (1993).

112. Toomay, John C. *Radar Principles for the Non-Specialist*. Lifetime Learning Publications, 1982.

113. Tung, Chi Tung and Kim Lin Chew. "The Multicriteria Pareto-optimal Path Algorithm," *Computers and Operations Research*, *62*(2):203–209 (1992).

114. Walker, Henry M. *Computer Science 2: Principles of Software Engineering, Data Types, and Algorithms*. Scott, Foresman and Company, 1989.

115. Wang, Chao-Chun and Leah H. Jamieson. "Autonomous Parallel Heuristic Combinatorial Search." *Seventh International Parallel Processing Symposium*. 741–746. April 1993.

116. Warren, C. W. "A Technique for Autonomous Underwater Vehicle Route Planning." *IEEE Symposium on Autonomous Underwater Vehicle Technology*. 201–205. 1990.

117. Warren, Charles W. "Fast Path Planning Using Modified A* Method." *IEEE International Conference on Robotics and Automation2*. 662–667. 1993.

118. Weiss, Mark Allen. *Data Structures and Algorithm Analysis*. Benjamin Cummings, 1992.

119. Wilfong, Gordon. "Shortest Paths for Autonomous Vehicles." *IEEE International Conference on Robotics and Automation,1*. 15–20. 1989.

120. Womble, David, et al. "102 Gflops on the Intel Paragon: How They Did It," *HPCwire* (March 1994).

121. Yen, I-Ling, et al. "Strategies for Mapping Lee's Maze Routing Algorithm onto Parallel Architectures." *IEEE Seventh International Parallel Processing Symposium*. 672–679. 1993.

Vita

Michael S. Gudaitis was born January 18, 1959, in New Britain, Connecticut. He enlisted in the Air Force in September, 1977, as a radio repairman. For the next eight years, he installed, repaired, and maintained radio equipment at Air Force Bases in Turkey, England, Canada, Panama, and the United States. In 1985, he was selected for the Airman's Education and Commissioning Program (AECP), and was sent to North Carolina State University to complete his undergraduate work in Electrical Engineering. He graduated magna cum laude, December, 1987, and was commissioned a second lieutenant, April, 1988, after three months of Officer Training School. Mike spent the next five years at Hanscom AFB, Bedford, Massachusetts, as a project manager and test engineer. He managed the acquisition and testing of a tactical, satellite communication system, and an air-to-ground secure data link for the Joint STARS aircraft. While at Hanscom, he completed a master's degree in electrical engineering at the University of Lowell (now called University of Massachusetts at Lowell). In May 1993, he was assigned to the School of Engineering, Air Force Institute of Technology, to complete a master of science degree in software engineering. His next assignment is at Rome Labs, RL/C3BB, Griffiss AFB, New York 13441-5700.

Michael Gudaitis is married to Jean, and has two children, Edward and Catherine.

Permanent address: 16 Welles Terrace
Meriden, CT 06450
(203) 237-8350

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE Multicriteria Mission Route Planning Using A Parallel A* Search	5. FUNDING NUMBERS
6. AUTHOR(S) Michael S. Gudaitis	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology WPAFB OH 45433-6583	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/94D-05
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Maj Dennis Andersh WL/AARA Wright-Patterson AFB, OH 45433	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release. Distribution Unlimite.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Mission Route Planning (MRP) Problem falls into the general class of multicriteria path search problems. Multiple criteria are evaluated to select an optimal aircraft mission route through a hostile environment. Criteria for distance travelled and radar exposure are combined into a single cost function for route evaluation. Radar calculations are performed dynamically. The A* search algorithm is applied to the MRP problem, and a parallel implementation is developed and tested. A unique combination of distributed OPEN lists with a global CLOSED list strategy produced fast execution times on the Paragon. Test cases for scenarios with 15 radars took less than 5 minutes with 16 processors. Measured performance is far superior to previous efforts.

14. SUBJECT TERMS Route Planning, A*, Best First, Monostatic Radar, Multiple Criteria, Path Planning	15. NUMBER OF PAGES 148
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL